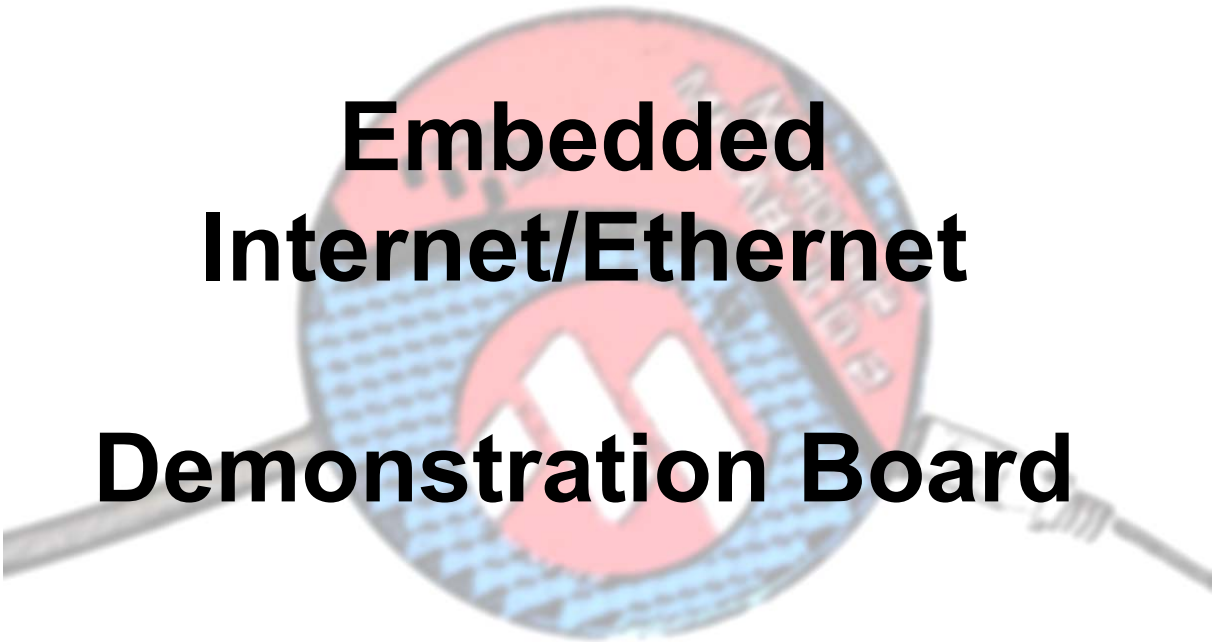


PICDEM.net™

Embedded Internet/Ethernet Demonstration Board



„Programmierung eines Microchip Controllers zur Steuerung eines Gleichstrom Motors mit PIC C18 (übers Netz).“

1	EINFÜHRUNG.....	- 4 -
2	GETTING STARTED.....	- 5 -
2.1	HARDWARE	- 5 -
2.1.1	<i>PICDEM.net Board.....</i>	- 5 -
2.1.2	<i>MPLAB ICD 2</i>	- 7 -
2.2	SOFTWARE.....	- 8 -
2.2.1	<i>MPLAB IDE v6.20</i>	- 8 -
2.3	PROGRAMMIERSPRACHEN.....	- 9 -
2.3.1	<i>C18 Compiler.....</i>	- 9 -
2.3.2	<i>Konfiguration der MPLAB IDE für C18</i>	- 9 -
2.4	PROGRAMMIEREN DER DEMO-FIRMWARE AUF EEPROM MIT MPLAB v6.20	- 9 -
2.4.1	<i>MPLAB DIE v6.20 Installation und Konfiguration.....</i>	- 10 -
2.4.2	<i>Firmware auf EEPROM schreiben</i>	- 11 -
2.5	DEMO FIRMWARE KONFIGURIEREN	- 12 -
3	UNTERSUCHEN VON HARDWARE UND PERIPHERIE DES PICDEM.NET DEMO BOARD- 15	
3.1	BIBLIOTHEKEN DES C18 COMPILER.....	- 15 -
3.1.1	<i>Aufteilung der Bibliotheken.....</i>	- 15 -
3.1.2	<i>Auflistung der unterstützten Bibliothek Routinen.....</i>	- 16 -
3.1.3	<i>Anpassen der Prozessor Spezifischen Bibliotheken für das PICDEM.net Board.....</i>	- 17 -
3.2	MCHPSTACK VON MICROCHIP – REVERSE ENGINEERING	- 18 -
3.2.1	<i>MCHPStack als einzige Quelle bei externe Peripherie.....</i>	- 18 -
3.2.2	<i>Durchsuchen des Quellcodes von MCHPStack am Beispiel der Port- und Pinbelegung des LCD-Display zur Ansteuerung</i>	- 18 -
3.2.3	<i>Ergebnis des „reverse Engineering“</i>	- 21 -
3.3	MICROCONTROLLER PIC18F452	- 21 -
3.3.1	<i>RISC (Reduced Instruction Set Computer) CPU.....</i>	- 21 -
3.3.2	<i>Peripherie.....</i>	- 22 -
3.3.3	<i>Analoge Eigenschaften.....</i>	- 22 -
3.3.4	<i>Spezielle Microcontroller Eigenschaften</i>	- 22 -
3.3.5	<i>Pin Diagram.....</i>	- 23 -
3.3.6	<i>Gerät Eigenschaften Überblick.....</i>	- 23 -
3.4	EXTERNE VERFÜGBARE PERIPHERIE	- 23 -
3.5	GENERELLE ERKENNTNIS / FAZIT	- 24 -
4	ANSPRECHEN DER PERIPHERIE UND HARDWARE IN C.....	- 25 -
4.1	ALLGEMEINE DEFINITIONEN - DEFINES.H	- 25 -
4.2	USER-LEDS.....	- 25 -
4.3	LCD DISPLAY.....	- 27 -
4.3.1	<i>Eigenschaften.....</i>	- 27 -
4.3.2	<i>LCD-Controller.....</i>	- 28 -
4.3.3	<i>Bibliotheksfunktionen.....</i>	- 32 -
4.3.4	<i>Ansteuerung in C.....</i>	- 33 -
4.4	SERIELLE SCHNITTSTELLE	- 33 -
4.4.1	<i>Eigenschaften.....</i>	- 33 -
4.4.2	<i>Register</i>	- 33 -
4.4.3	<i>Konfigurieren des USART in C</i>	- 37 -
4.5	AD-WANDLER.....	- 37 -
4.5.1	<i>Eigenschaften.....</i>	- 37 -
4.5.2	<i>Register Konfiguration.....</i>	- 37 -
4.5.3	<i>Ansteuerung in C.....</i>	- 40 -
4.6	INTERRUPTS.....	- 43 -
4.6.1	<i>Eigenschaften.....</i>	- 43 -
4.6.2	<i>Implementieren einer Interrupt Service Routine in C</i>	- 44 -
5	BEISPIELPROGRAMME – TUTORIALS.....	- 46 -
5.1	GRUNDSÄTZLICHES	- 46 -
5.2	AD-WANDLER UND ANZEIGE AUF DEM LCD-DISPLAY	- 46 -

5.2.1	Beschreibung.....	- 46 -
5.2.2	Projekt erstellen und konfigurieren.....	- 47 -
5.2.3	Source Code	- 51 -
5.2.4	Programmieren des Microcontroller mit ICD 2.....	- 54 -
5.2.5	In-Circuit-Debuggen mit dem ICD 2.....	- 56 -
5.3	ZEICHEN ÜBER SERIELLE SCHNITTSTELLE UND ANZEIGE AUF LCD.....	- 60 -
5.4	TIMER INTERRUPTS UND SERIELLE SCHNITTSTELLE INTERRUPT	- 60 -
6	PROJEKTAUFGABE UND DURCHFÜHRUNG.....	- 61 -
6.1	EINFÜHRUNG IN DAS PROJEKT	- 61 -
6.1.1	Aufgabenstellung.....	- 61 -
6.1.2	Verwendete Materialien	- 61 -
6.1.3	Verlagerung des Projektschwerpunkts.....	- 61 -
6.2	REALISIERUNG HARDWARE.....	- 62 -
6.2.1	Einführung	- 62 -
6.2.2	Allegro SA3968A H-Brücke	- 62 -
6.2.3	Motor.....	- 63 -
6.2.4	Externe Schaltung	- 64 -
6.3	REALISIERUNG SOFTWARE	- 66 -
6.3.1	Ansteuerung des Motors über die serielle Schnittstelle.....	- 66 -
6.3.2	Motoransteuerung per Ethernet.....	- 68 -
6.4	FAZIT	- 71 -
ANHANG.....		- 72 -
6.5	DATENBLÄTTER/QUELLEN	- 72 -
6.6	ORDNERSTRUKTUR DER CD	- 72 -
6.7	SOURCE CODE MOTOR_UDP.....	- 73 -
6.7.1	motor_server.h.....	- 73 -
6.7.2	motor_server.c	- 74 -
6.7.3	StackTsk.h	- 79 -
6.7.4	decoder.h.....	- 84 -
6.7.5	decoder.c	- 84 -
6.7.6	interrupt.h	- 85 -
6.7.7	interrupt.c.....	- 85 -
6.7.8	websrvr.c.....	- 86 -
6.8	SOURCE CODE MOTOR_SERIAL.....	- 100 -
6.8.1	usart.h	- 100 -
6.8.2	usart.c.....	- 100 -
6.8.3	xlcd.h.....	- 101 -
6.8.4	xlcd.c	- 103 -
6.8.5	defines.h	- 107 -
6.8.6	decoder.h	- 108 -
6.8.7	decoder.c.....	- 108 -
6.8.8	interrupt.h	- 109 -
6.8.9	interrupt.c.....	- 109 -
6.8.10	motor_serial.c.....	- 110 -

1 Einführung

Das gesamte Projekt lief unter der Aufsicht von Prof. Dr. Linn und der Lehrveranstaltung PDV-Vertiefung. Zur genauen Projektbeschreibung und Durchführung sehe man im Kapitel 5 nach. Die vorherigen Kapitel geben eine Einführung in die Materie da dies doch ein weitaus größeres Problem darstellte als man sich vorher gedacht hat.

Das erhaltene PICDEM.net Board hat die besondere Eigenschaft einen embedded Webserver zu ermöglichen. Ein RJ-45 Anschluss ist verfügbar um das Board an das Ethernet anzuschließen.

Eine frei verfügbarer, mitgelieferter TCT/IP Stack kann für eigene Zwecke verwendet werden. Zudem ist eine Testsoftware, die schon vorinstalliert ist, verfügbar um alle Feinheiten des Boards zu testen.

Die Anwendung für das Board bezieht sich nun nicht mehr ausschließlich dem Ansprechen, durch eigens dafür geschriebene Software, einzelner Peripherie sondern diese können nun auch über eine Webseite angesprochen werden, die direkt auf dem Board gespeichert wird oder über ein eigenes Protokoll welches über Ethernet läuft..

2 Getting Started

Das erhaltene Board hat die Testsoftware schon vorinstalliert. Als erstes wollen wir versuchen „einfach nur“ über das Netzwerk auf die Webseite zuzugreifen. Da Default mäßig die Ethernet Konfiguration auf DHCP eingestellt ist und wir aber eine statische IP vergeben wollen ist die erste Berührung mit dem Board die Konfiguration der IP.

Das Board kann man über die serielle Schnittstelle mit einem Nullmodem Kabel, dem installierten Konfigurationsmenü und einem Terminal Programm konfigurieren. In dem mitgeliefertem Einführungsmanual werden die Einstellungen des Hyper-Terminals von Windows sowie die ganzen mitgelieferten Teile sehr gut erklärt.

Da wir festgestellt haben, dass die schon installierte Firmware fehlerhaft war musste erst eine aktuellere Version auf das EEPROM geschrieben werden. Daher wird das Programmieren des EEPROMs vor dem Konfigurieren der Firmware behandelt.

Zunächst aber erst eine Auflistung und Erläuterung aller verwendeten Geräte und der Software.

2.1 Hardware

2.1.1 PICDEM.net Board

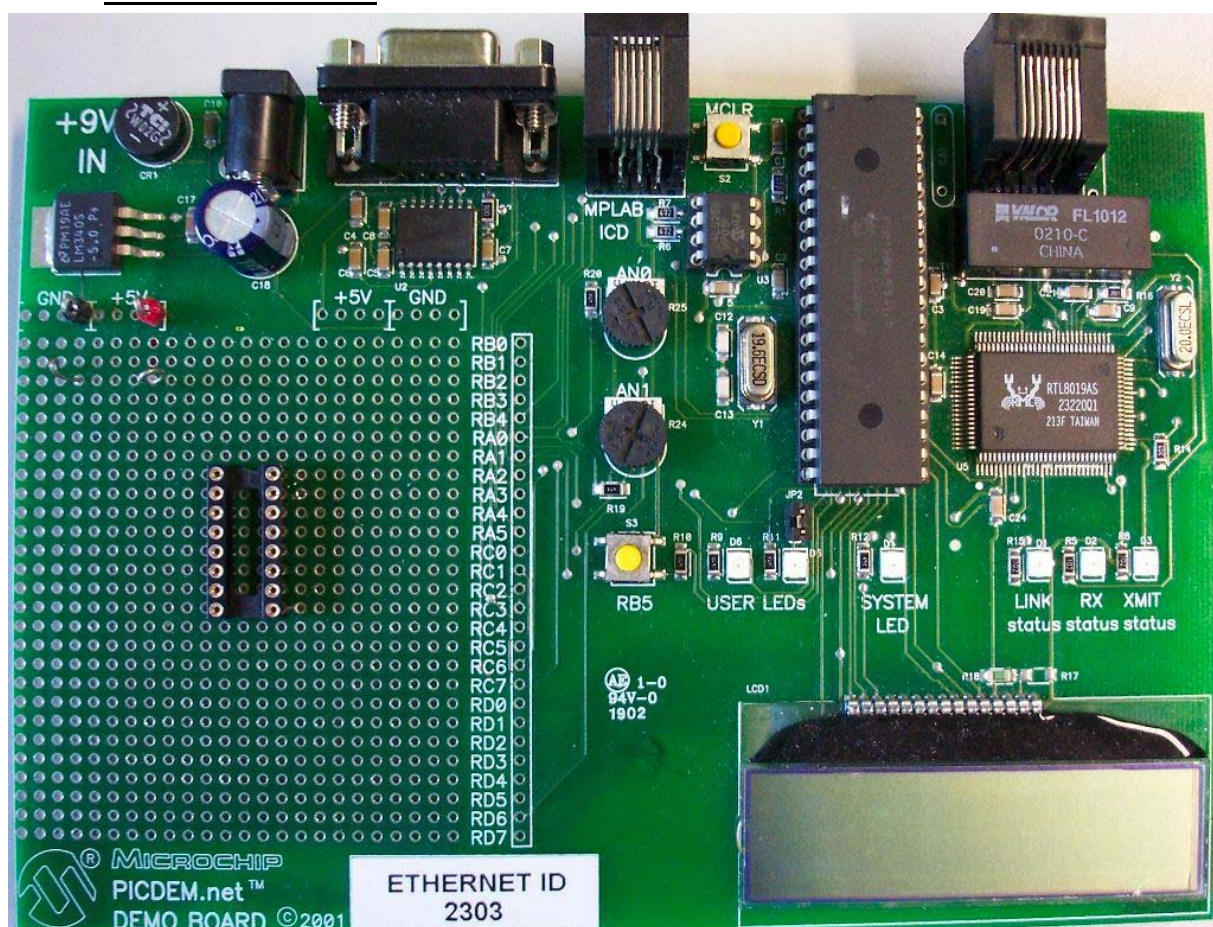


Bild 1 - PICDEM.net Demo Board

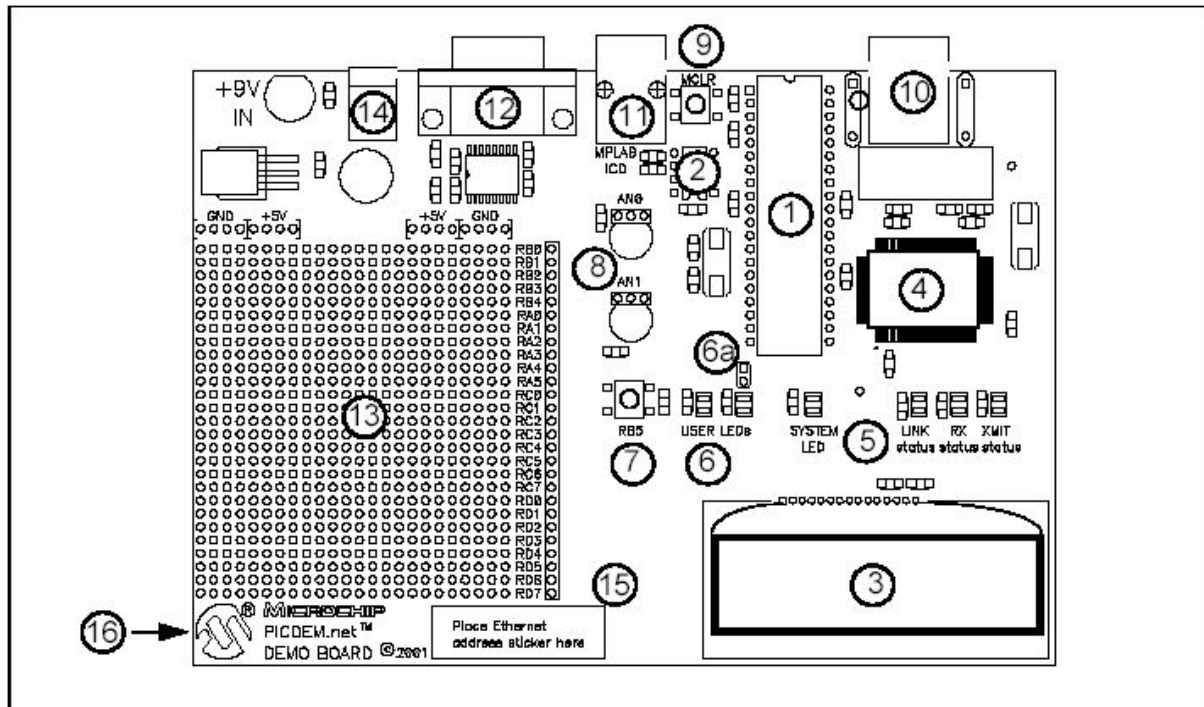


Bild 2 Bestückung PICDEM.net Board

Nr.	Erläuterung
1	MICROCONTROLLER SOCKEL (40 Pins): Dieses Board unterstützt die gängigen Microchip PICMicro Microcontroller. Fabrikneu enthält dieses Board den PIC 18F452 Microcontroller welcher mit einer Clockfrequenz von 19.6608 MHz läuft. Zusätzlich ist der Microcontroller mit der Demo Applikation welches den TCP/IP Stack von Microchip verwendet vorprogrammiert.
2	SPEICHER CHIP: Ein EEPROM welches 256 KBit speichern kann (32 KB). Hier können für den embedded Webserver die html Seiten abgespeichert werden.
3	LCD DISPLAY: Ein 2 Zeilen LCD Display zur visuellen Ausgabe von Status- oder Fehlermeldungen.
4	ETHERNET CONTROLER: Es wird der Realtek RTL8019AS single Chip genutzt.
5	STATUS LEDs: Es werden 4 LEDs unterstützt: • SYSTEM: Leuchtet wenn das Board die Stromversorgung erhält und richtig verbunden ist. • LINK STATUS: LED leuchtet wenn eine Verbindung besteht. • XMIT und RX: LEDs blinken (invertierte Logik) wenn das Board Pakete versendet oder empfängt.
6	BENUTZERDEFINIERTER LEDs: Diese beiden LEDs werden von einem digitalen Output des Controller gesteuert. Sie können benutzt werden um einen digitalen Output zu simulieren. Beide LEDs können durch einen Jumper deaktiviert werden (Position 6a).

7	BENUTZERDEFINIERTER PUSH BUTTON: Dieser Schalter ist mit einem Digitalen Eingang des Controller verbunden.
8	BENUTZERDEFINIERTER POTENTIOMETER: Zwei 10 kOhm Potentiometer die mit 2 analogen Eingängen des Controller verbunden sind. Die Bezeichnungen AN0 und AN1 sind auf der Platine vertauscht.
9	RESET PUSH BUTON: Dieser Schalter ist fest mit dem MCLR Pin des Controller verbunden.
10	RJ-45 (10-Base T) DOSE: Port für die Standart Ethernet Verbindung.
11	RJ-11 (SIX-WIRE) DOSE: Dieser Port ermöglicht es mit dem MPLAB ICD System zu arbeiten welches das onboard programmieren des EEPROMS erlaubt. Zusätzlich lässt sich damit auch debuggen.
12	RS-232 PORT: Ermöglicht eine Verbindung über die Serielle Schnittstelle.
13	PROTOTYP BEREICH: Ein 24x27 breites Lochfeld welches für eigene Schaltungen und Erweiterungen zur Verfügung steht. Es gibt eine Spannungsversorgung von 5 VDC und es können verschieden Ports des Controllers angesprochen werden. (RA<5:0>, RB<4:0>, RC<7:0> und RD<7:0>).
14	SPANNUNGSVERSORGUNG:
15	ETHERNET ID: Diese einmalige Seriennummer repräsentiert die letzten beiden Bytes der vorprogrammierten Media Access Control Nummer (MAC Adresse).
16	

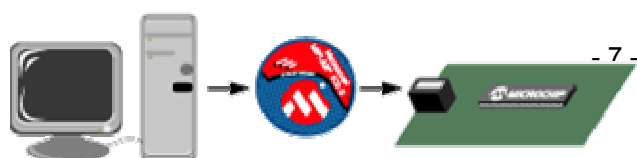
2.1.2 MPLAB ICD 2



Der MPLAB ICD 2 ist ein preisgünstiger Echtzeit Debugger und Programmierer für ausgewählte PICmicro Microcontroller.

Grundsätzlich kann man zwei große Vorteile diesem Gerät und der Technologie nachsagen. Der erste ist das Programmieren des Controllers während dieser noch auf dem Board sitzt. Er muss also nicht mehr, wie noch bei vielen Projekten üblich und bekannt, immer vom Sockel geholt und auf ein entsprechendes Programmiergerät gesteckt werden. Der zweite große Vorteil ist das „In-Circuit Debuggen“. Es ist Möglich die Werte von ausgewählten Variablen zur Laufzeit zu beobachten oder sich

Breakpoints setzen. Zudem kann man noch jede Codezeile einzeln in Schritten durchgehen.



Das Gerät und die Technologie in Zusammenhang mit der Software MPLAB IDE bringen komfortable Vorteile mit in die Entwicklung von Embedded Systems. Erwähnenswert an dieser Stelle ist, wenn man MPLAB ICD 2 mit der USB Schnittstelle benutzen möchte sollte man die MPLAB IDE Software vorher installieren denn sonst kann es passieren, dass das Gerät nicht korrekt gefunden wird.

Bei der Installation der Software wird am Ende auch eine Installationsanleitung für MPLAB ICD 2 angezeigt welcher man auch genauestens folgen sollte.

Für aktuelle Informationen über MPLAB ICD 2 kann auf der Homepage von Microchip geschaut werden.

<http://www.microchip.com/1010/pline/tools/picmicro/icds/icd2/>

2.2 Software

2.2.1 MPLAB IDE v6.20

Die IDE (Integrated Development Environment) die Verwendet wird ist die von Microchip entwickelte MPLAB IDE v6.20. Sie ermöglicht es Projekte zu managen, zu programmieren oder debuggen. Unterstützt werden 2 Programmiersprachen – Assembler und C18 (eine abgespeckte C Variante – mehr dazu in Kapitel 2.3).

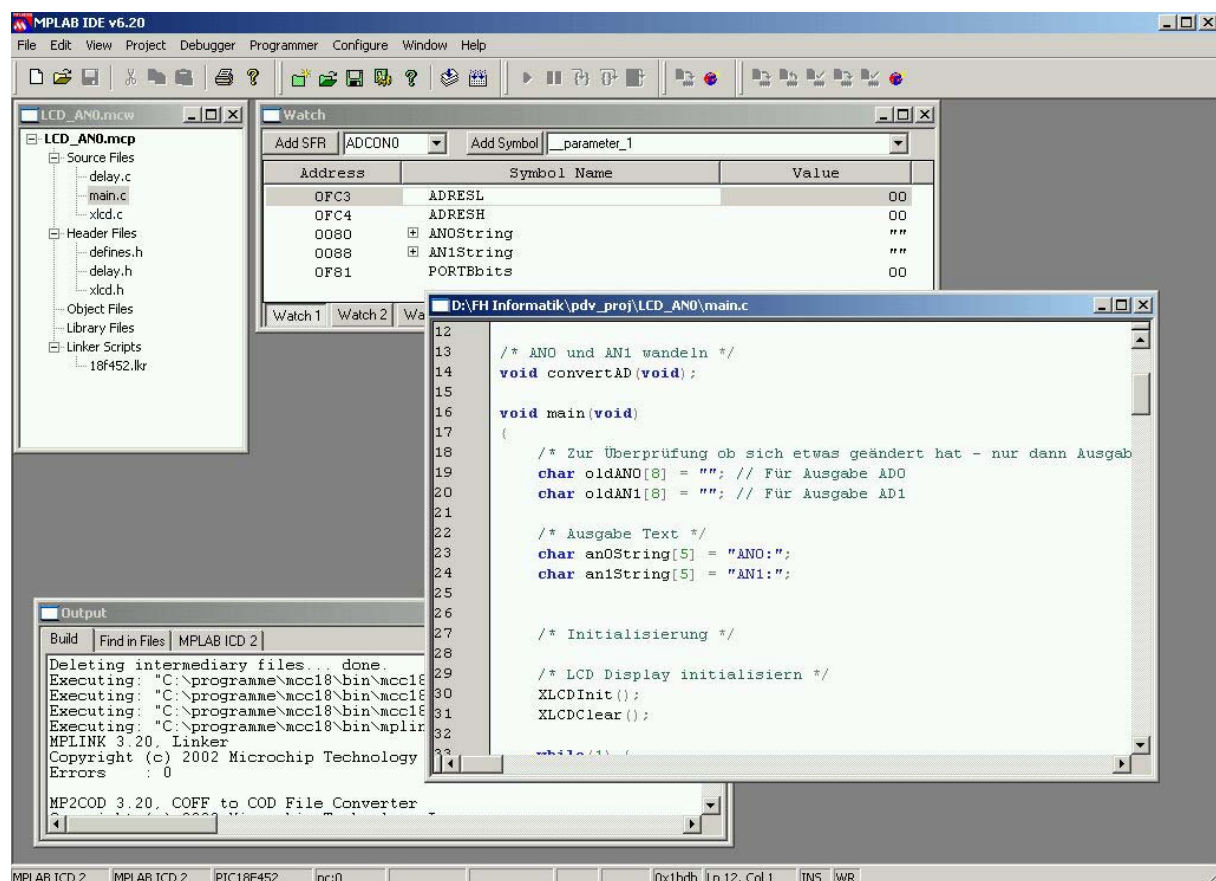


Bild 3 MPLAB IDE v6.2 Entwicklungsumgebung

Auf dem Bild kann man den Desktop der Entwicklungsumgebung mit einzelnen Fenstern erkennen. Unter anderem ist links oben das Projektfenster, welches alle Projektdateien hält. Rechts daneben ist die „Watch“ welches beim Debuggen verschiedene ausgewählte Variablen mit dessen Werten anzeigt. Dann ist da noch

ein normales Editor Fenster und unten links das Output Fenster, welches die ganzen Status oder Compiler Meldungen anzeigt.

Es empfiehlt sich immer die neueste Version zu nehmen denn eines unserer Anfangsprobleme war, dass es nur v6.10 gab und diese nicht fast USB 2.0 unterstützte, wie sich später herausstellte. Das hatte dann längere Zeit die Folge, dass das ICD 2 nicht erkannt wurde und es Merkwürdige und nicht Erkennbare Fehler beim Programmieren des Microcontrollers gab. Erst mit der Version 6.20 lief es einwandfrei.

Die neueste Version kann man auf der Homepage von Microchip runterladen.

<http://www.microchip.com/1010/pline/tools/picmicro/devENV/mplabi/mplab6/index.htm>

Zudem sollte noch Erwähnt werden, dass das Verzeichnis in dem die Projekte liegen die Länge von 64 Zeichen nicht überschreiten darf denn sonst bricht der Compiler ab und es kann nicht kompiliert werden.

2.3 Programmiersprachen

Es gibt die klassische Methode die ganze Entwicklung in Assembler zu schreiben wie es bestimmt auch noch in vielen Projekten getan wird. Wenn man dies tun möchte kann man die Entsprechende Dokumentation dazu lesen die in Mengen vorhanden ist.

Wir haben uns allerdings bei unserem Projekt dafür entschieden eine Hochsprache zu verwenden welches eine Abgespeckte C Version ist. Daher wird hier nur eine Einführung für den C18 Compiler und nicht für Assembler gegeben.

2.3.1 C18 Compiler

Der MPLAB C18 Compiler ist ein freier, optimierter ANSI C Compiler für die PIC18 PICmicro Microcontroller (MCU). Der Compiler ist eine 32-Bit Windows Konsolen Applikation und voll Kompatibel mit der MPLAB IDE von Microchip. Er ermöglicht zudem auf Quellcodeebene das Debugging mit dem MPLAB ICD2 In-Circuit Debugger.

Eine genauere Beschreibung und eine Tiefergehende Einführung in den C18 Compiler und den Bibliotheken finden Sie in Kapitel 3.

Aktuelle Compiler Versionen sowie Datenblätter und User Guides kann man von der Homepage von Microchip runterladen.

<http://www.microchip.com/1010/pline/tools/picmicro/code/mplab18/>

2.3.2 Konfiguration der MPLAB IDE für C18

Der Compiler ist voll Kompatibel zur MPLAB IDE allerdings müssen vor der Nutzung noch einige Einstellungen getätigt werden wenn man in C programmieren möchte.

2.4 Programmieren der Demo-Firmware auf EEPROM mit MPLAB v6.20

Zu dem Demo Board wird der In-Circuit Debugger – MPLAB ICD 2 vom Microchip verwendet. Das ermöglicht uns das direkte beschreiben des EEPROMs ohne ihn vom Board abnehmen zu müssen.

2.4.1 MPLAB DIE v6.20 Installation und Konfiguration

Es wird als Entwicklungsumgebung die Software MPLAB IDE v6.20 von Microchip verwendet, die man direkt von der Homepage herunterladen kann.

Diese Software ermöglicht die Entwicklung von Projekten in verschiedenen Sprachen (z.B. Assembler oder C). Zudem kann man das Projekt Simulieren und Debuggen. Nach der Installation ist wichtig, dass man den Treiber des ICD 2 Geräts für die USB Schnittstelle korrekt installiert. Es gehen direkt nach der Installation verschiedene Browser mit der Installationsbeschreibung auf, da ein eigener mitgelieferter Treiber verwendet wird und Windows das Gerät nicht automatisch erkennt.

Die korrekte Installation der Hardware-Geräte wird vorausgesetzt.

Die erste Einstellung, bevor man ein Projekt bearbeitet, ist das Gerät was benutzt wird – in unserem Fall der PIC18F452.

Erreichbar im Menü: Configure->Select Device...

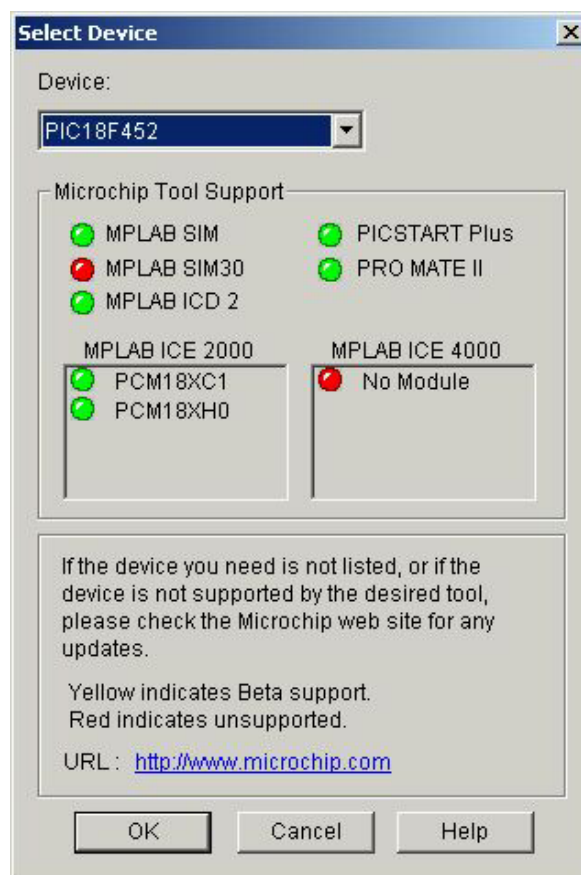


Bild 4 Configure->Select Device

Im Kästchen Microchip Tool Support kann man erkennen welche Tools von der Software unterstützt werden. Zur Simulation wird MPLAB SIM benötigt und zum Beschreiben des EEPROM MPLAB ICD 2.

2.4.2 Firmware auf EEPROM schreiben

Als erstes muss man Angeben welches Gerät zur Programmieren verwendet wird.
Programmer->Select Programmer->MPLAB ICD 2

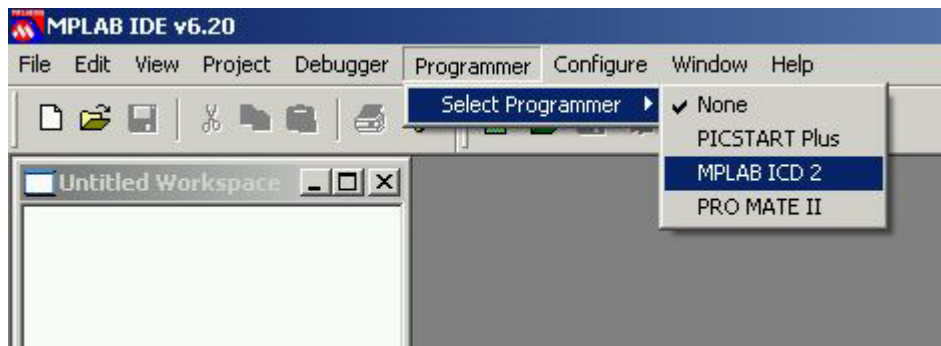


Bild 5 Programmer->Select Programmer

Es öffnet sich direkt danach ein Output Fenster welches den Status des Connect-Tests wiedergibt. Wenn alle Geräte korrekt angeschlossen sind, sollte folgender Output zu sehen sein:

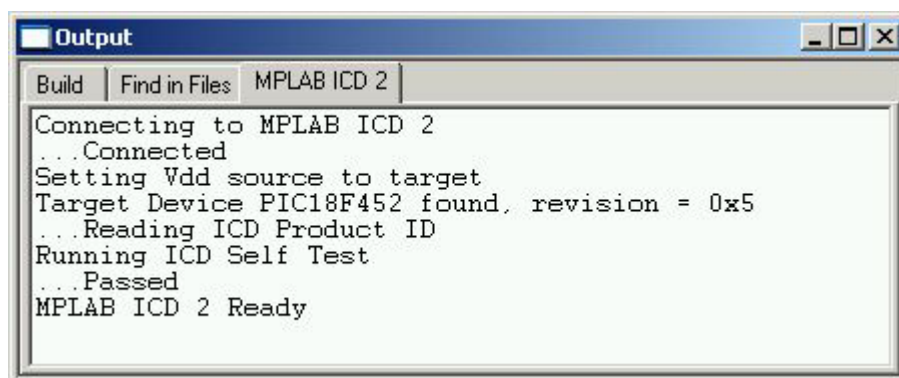


Bild 6 Select Device - Output

Zudem öffnet sich noch eine Icon-Menüleiste mit alle Funktionen welche für das Programmieren des Targets notwendig sind (z.B. Program target device oder Reset and Connect to ICD)

Da die mitgelieferte Firmware schon in einem vorkompiliertem HEX-File vorhanden ist wird kein neues Projekt erstellt sondern das File wird Importiert (File->Import...). Im Anschluss kann man leider nirgends erkennen, dass etwas importiert wurde.

Für die Firmware wurden folgende Configuration Bits vorgegeben:

- Oscillator: HS
- Watchdog Timer: Disabled
- Low Voltage Programming: Disabled

Diese kann man im Menü unter Configure->Configuration Bits... finden.

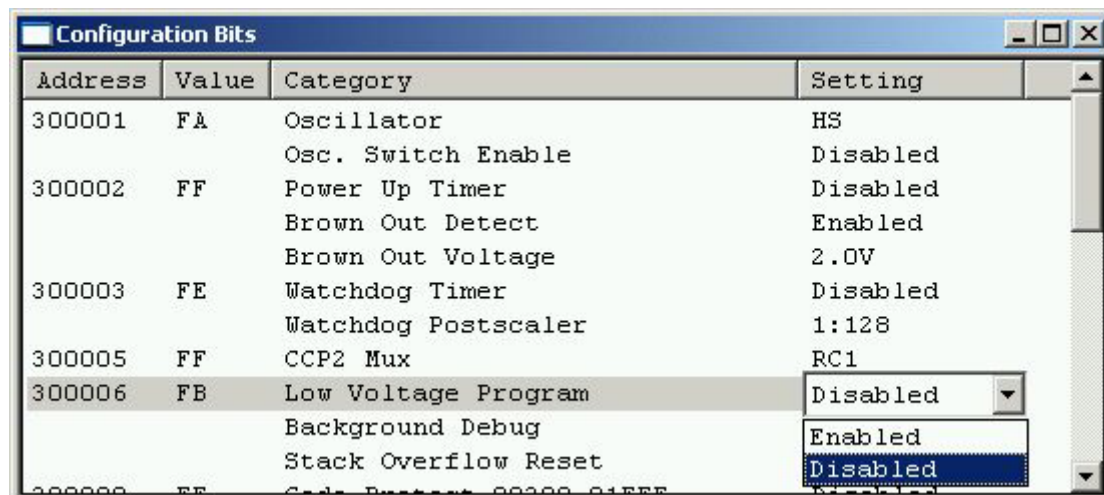


Bild 7 Configurations->Configuration Bits

Wenn nun alle Einstellungen getätigt sind, kann man das importierte HEX-File (Firmware) auf das EEPROM programmieren.

Programmer->Program oder auf das erste Icon (Program target device) klicken.
Je nach dem wie groß das HEX-File ist, umso länger kann das downloaden dauern.
Nach dem korrekten download sollte folgende Meldung im Output Fenster erscheinen:

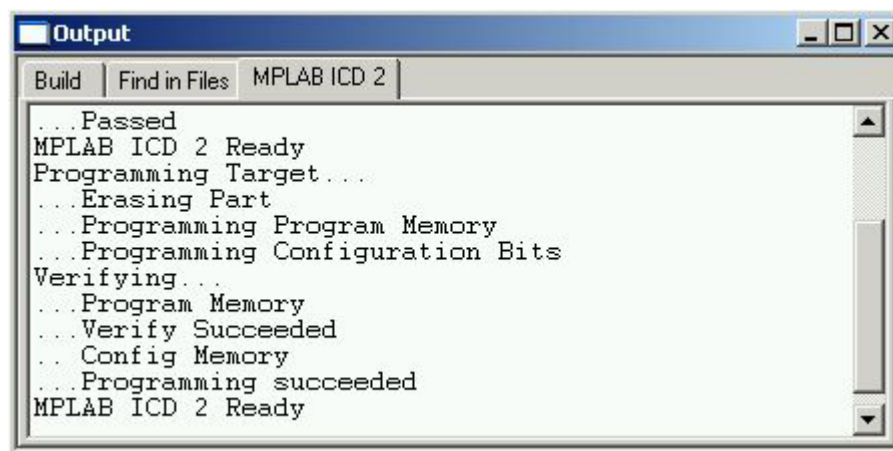


Bild 8 Program EEPROM Output

Um das Programm auf dem Demo Board zu testen muss nur noch das Kabel von dem ICD 2 Gerät zu dem Board entfernt werden da sonst das Board nicht reagiert.

2.5 Demo Firmware konfigurieren

Da die Firmware default mäßig auf DHCP eingestellt ist muss sie erst konfiguriert werden. Dies geschieht mit einem einfachen Terminal Programm (z.B. Hyper-Terminal von Microsoft).

Die Einstellungen für das Terminalprogramm sind folgende:

- Bits per Second: 19200
- Data Bits: 8
- Parity: none
- Stop Bits: 1
- Flow Control: none

Nachdem das Terminal konfiguriert ist kann man durch gleichzeitiges drücken beider Taster das Setup Menü aufrufen. Auf dem LCD Display des Demo Boards erscheint die Schrift „Board Setup...“ und im Hyper-Terminal folgendes:

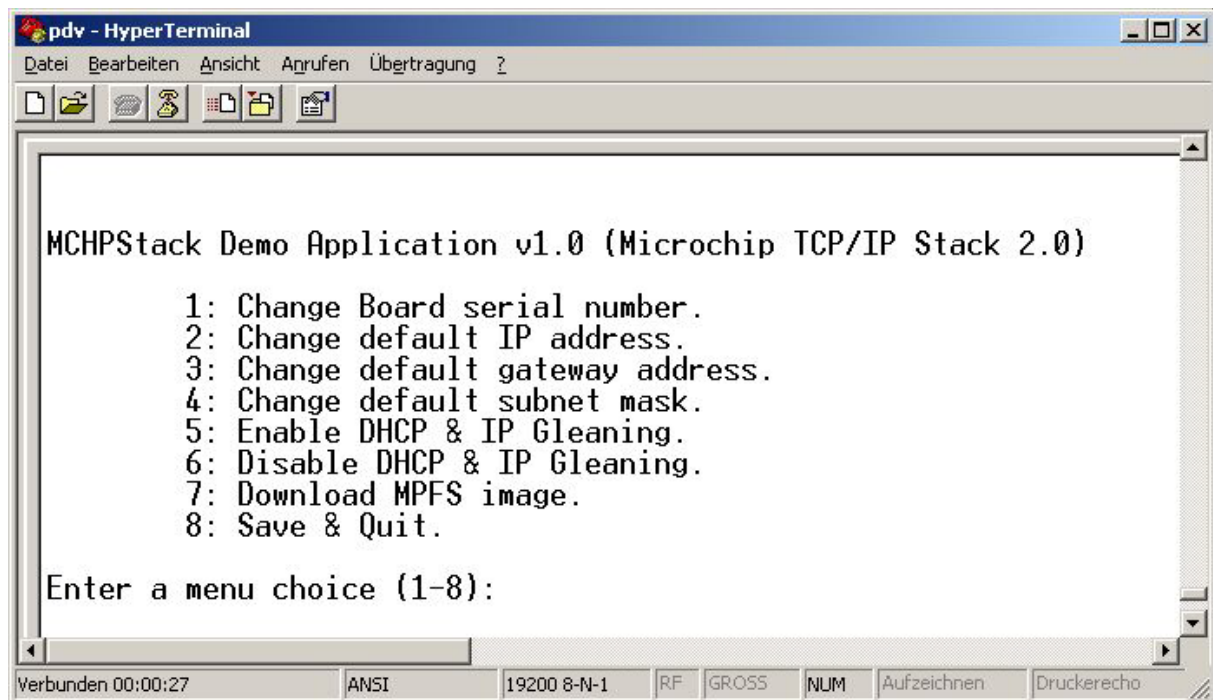


Bild 9 Hyper-Terminal Konfiguration Firmware

Um die IP-Adresse zu ändern muss nun die 2 gedrückt werden. Nach der Eingabe der neuen Adresse kann bei Bedarf auch noch die Subnet Mask geändert werden. Als nächstes wird noch Menüpunkt 6 gewählt um DHCP auszuschalten. Zum Schluss die Konfiguration durch drücken der 8 beenden.

Wenn man nun die eben bei der Konfiguration angegebene IP-Adresse in den Browser eingibt sollt die Firmware-Homepage erscheinen, sofern das Demo Board auch mit im gleichen Ethernet hängt.

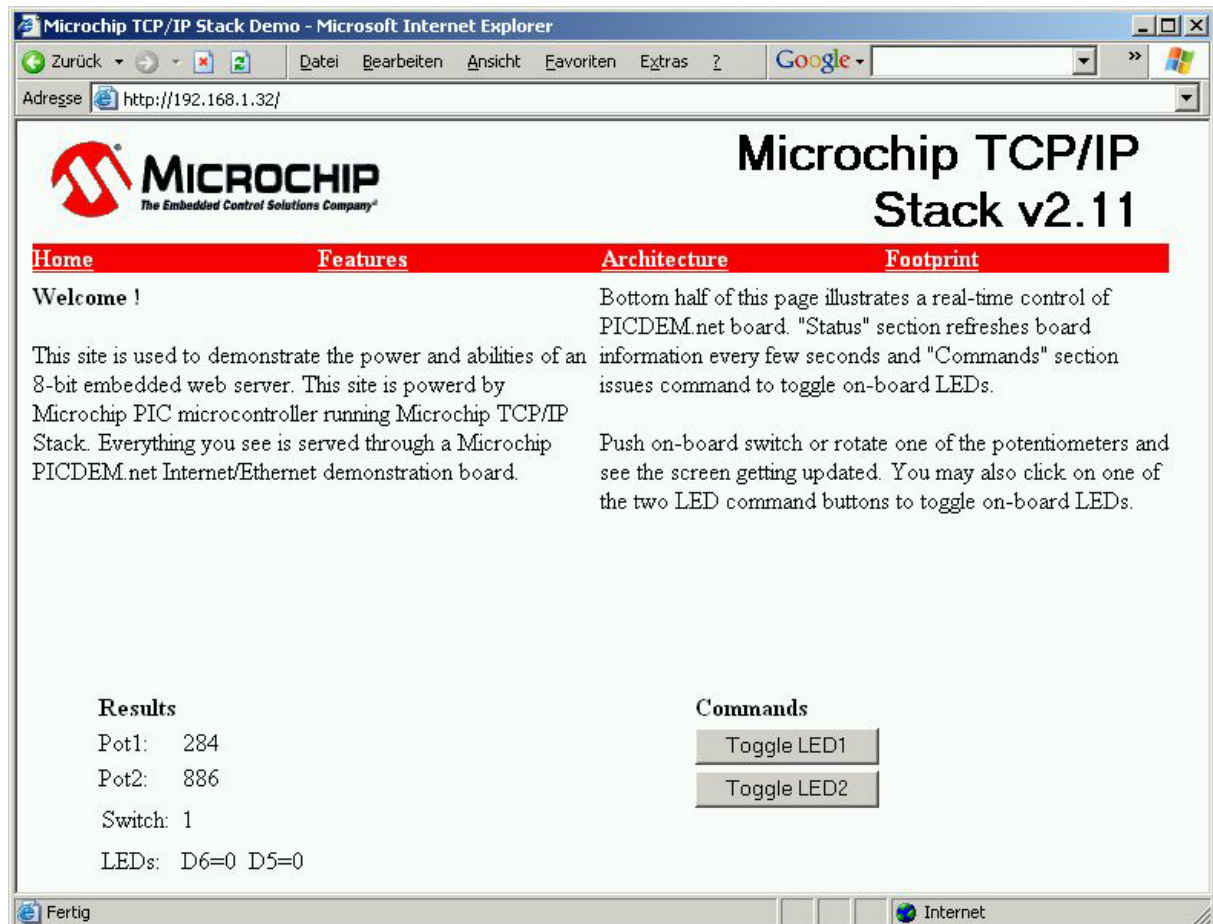


Bild 10 Firmware Homepage auf Demo Board

3 Untersuchen von Hardware und Peripherie des PICDEM.net Demo Board

Als kleines Vorwort wollen wir hierzu sagen, dass wir keine Dokumentation zur Bestückung des PICDEM.net Board und dessen Port- und Pinbelegung im Internet oder bei dem Hersteller selbst gefunden haben. Außer einer kleinen Einführung bei der grob aufgelistet wurde an welcher Stelle sich auf dem PICDEM.net Board welche Peripherie befindet kann man nichts finden was auf entsprechende Ansteuerung der Peripherie hinweist.

Da unser für unser Projekt unter anderem eine lokale Ausgabe auf dem LCD Display gewünscht ist war die erste Teilaufgabe das LCD-Display, welches eine externe Peripherie auf dem PICDEM.net Board ist, anzusprechen. Mit diesem Ansatz ist es zwangsweise zu diesem Kapitel mit der allgemeinen Überschrift „Untersuchen des PICDEM.net Board“ gekommen und haben bemerkt, dass es durch nicht vorhandene Dokumentation über Port- und Pinbelegung sowie der einzelnen Bauteile sehr Mühsam ist sich das zu erarbeiten.

Daher beginnt dieses Kapitel erst mit einer kleinen Erklärung des C18 Compilers, da dieser laut Dokumentation Bibliotheken mitbringt die für das Ansprechen der Peripherie genutzt werden können.

Zudem soll gezeigt werden wie wir vorangegangen sind und woher wir unsere Informationen bezogen haben bevor wir uns dem wirklichen Ansprechen der Peripherie zuwenden (nächstes Kapitel). Das Problem der nicht Vorhandenen Dokumentation ist mit einer der Gründe dafür, dass wir in der Zeit soweit zurückgeworfen wurden. Denn normalerweise sollte man von einem Hersteller erwarten können, dass er zu seinem Produkt welches er verkauft auch eine entsprechende Dokumentation mitliefert.

3.1 Bibliotheken des C18 Compiler

3.1.1 Aufteilung der Bibliotheken

Der Compiler hält sich, wie schon beschrieben, an ANSI C Standart 1989 was auch für seine mitgelieferten Bibliotheken gilt. Die Bibliotheken kann man in 4 Gruppen fassen.

- Hardware Peripherie Funktionen
- Software Peripherie Bibliothek
- Allgemeine Software Bibliothek
- Mathe Bibliothek

Hierbei wird jetzt noch einmal unterschieden zwischen Prozessor-spezifischen und allgemeinen Bibliotheken. Die „Hardware Peripherie Funktionen“ und die „Software Peripherie Bibliothek“ sind Prozessor spezifisch denn sie beinhalten alle Peripherie Funktionen und die „special function register“ (SFR) Definitionen.

Die Prozessor-spezifischen Bibliotheken sind nach folgendem Schema benannt:

 p *processor* .lib - ohne Leerzeichen natürlich

Für unseren verwendeten Mikroprozessor, den 18f452 wurde also immer die Bibliothek *p18f452.lib* verwendet.

Die „Allgemeine Software Bibliothek“ und die „Mathe Bibliothek“ gelten für die gesamte PIC18 Architektur und somit ist auch keine Prozessorabhängige Unterscheidung notwendig. Wenn Funktionen aus diesen beiden Bibliotheken verwendet wurden so wird die *clib.lib* hinzugelinkt.

Alle verwendeten Bibliotheken kann man für sich selber umschreiben oder anpassen. Die dazugehörigen Source-Files findet man im Verzeichnis: `/install_directory/src/`. Wenn nun einige Funktionen umgeschrieben wurden so muss die Bibliothek neu erstellt werden. Dazu liegen im `/src/` Verzeichnis verschiedene Batch Dateien die das erstellen übernehmen. Eine genaue Beschreibung kann man der MPLAB-C18-Libraries.pdf Dokumentation entnehmen.

Im eigenen Source-Code müssen natürlich die entsprechenden Header-Files inkludiert werden damit die Prototypen der Bibliotheks-Funktionen bekannt sind. Für die Einbindung der spezifischen Mikroprozessorbibliothek reicht der folgende include:

```
#include <p18cxxx.h>
```

Dadurch, dass man den genauen Mikroprozessortyp in der Projektumgebung angegeben hat wird die richtige Bibliothek automatisch bei dem kompilieren dazu gelinkt. Man kann aber auch gleich den richtigen Mikroprozessor angeben – das macht keinen Unterschied:

```
#include <p18f452.h>
```

Die allgemeine Variante ist sinnvoll wenn auf dem gleichen Board einen anderen Mikroprozessor benutzen möchte. Dadurch muss man nicht mehr im gesamten Code sondern nur in der Projektumgebung einmal den Mikroprozessortyp ändern.

3.1.2 Auflistung der unterstützten Bibliothek Routinen

Eine genaue Auflistung aller Funktionen, deren Parameter und Rückgabewerte werden ausführlich in der mitgelieferten Dokumentation des C18 Compilers beschrieben (MPLAB-C18-Libraries.pdf).

Hier ist nur eine übersichtliche Auflistung aller Funktionsgruppen. Man kann so schnell einen Eindruck über die implementierten Funktionen gewinnen.

Hardware Peripherie Funktionen

- A/D Converter Funktionen
- Input Capture Funktionen
- I²C Funktionen
- I/O Ports Funktionen
- Microwire Funktionen
- Pulsweiten Modulation Funktionen
- SPI Funktionen
- Timer Funktionen
- USART Funktionen

Software Peripherie Bibliothek

External LCD Funktionen
External CAN 2510 Funktionen
Software I²C Funktionen
Software SPI Funktionen
Software UART Funktionen

Allgemeine Software Bibliothek

Zeichen Klassifizierungs-Funktionen
Daten Konvertierungs-Funktionen
Delay Funktionen
Speicher und String Manipulations-Funktionen

Mathe Bibliothek

Für eine genaue Erklärung wird hier auf das „Embedded Control Handbook, Volume 2“ hingewiesen.
Des weiteren gibt es eine Erklärung wie die 32-Bit Floating Point Variablen aufgebaut und konvertiert werden. Eine Einführung gibt es im Kapitel 5 des MPLAB-C18-Libraries.pdf Dokument.

3.1.3 Anpassen der Prozessor Spezifischen Bibliotheken für das PICDEM.net Board

Nachdem nun herausgefunden wurde, dass es eine eigene Bibliothek für unseren verwendeten Microcontroller gibt und wir nur noch dessen Funktionen benutzen und verstehen müssen war ein gewisser Optimismus bei uns zu Erkennen welcher auch gleich wieder getrübt werden sollte.

Es gibt ein wunderbares Tutorial (MPLAB-C18-Getting-Started.pdf) von Microchip welches eine Einführung mit Beispielen für die Verwendung des C18 Compilers und dessen Peripherie in C und mit der MPLAB IDE v6.xx beschreibt. Leider gelten diese Beispiele wieder nicht für unser erhaltenes PICDEM.net Demo Board. Es wird das PICDEM 2 Plus Demo Board verwendet. Dieses wird auch mit dem Mikrokontroller 18f452 betrieben. Daher waren die verwendeten Beispiele nicht wirklich verwendbar sondern nur zur kurzen Anschauung Sinnvoll.

Jetzt kommen wir auch schon zum Eigentlichen Problem in dieser Phase des Projekts. Da nicht nur die Beispiele sondern auch die ganzen „Software Peripherie Bibliothek“ für das PICDEM 2 Plus Demo Board ausgelegt war konnte man wichtige Teile der Bibliothek nicht verwenden. Also alle Funktionen die z.B. für das Ansprechen der Seriellen Schnittstelle, Dioden oder des LCD-Displays zur Verfügung standen konnten auf Anhieb nicht genutzt werden.

Da dies ja im Eigentlichen Sinn kein Problem darstellen sollte wurden die Header-Files der einzelnen Source Codes im Verzeichnis /install_directory/h/ für die Anpassung auf das verwendete Board zur Verfügung gestellt.

Hierfür ein Auszug aus dem XLCD-Header File:

```
/* DATA PORT defines the port to which the LCD data lines are connected */  
#if 18CXX  
#define DATA_PORT PORTB  
#define TRIS_DATA_PORT TRISB  
#else /* 17CXX */  
#define DATA_PORT PORTC
```

```

#define TRIS DATA PORT DDRC
#endif

/* CTRL PORT defines the port where the control lines are connected.
 * These are just samples, change to match your application.
 */
#if 18CXX
#define RW_PIN PORTBbits.RB6 /* PORT for RW */
#define TRIS_RW DDRBbits.RB6 /* TRIS for RW */
#define RS_PIN PORTBbits.RB5 /* PORT for RS */
#define TRIS_RS DDRBbits.RB5 /* TRIS for RS */
#define E_PIN PORTBbits.RB4 /* PORT for E */
#define TRIS_E DDRBbits.RB4 /* TRIS for E */
#else /* 17CXX */
#define RW_PIN PORTCbits.RC5 /* Port for RW */
#define TRIS_RW DDRCbits.RC5 /* TRIS for RW */
#define RS_PIN PORTCbits.RC4 /* Port for RS */
#define TRIS_RS DDRCbits.RC4 /* TRIS for RS */
#define E_PIN PORTCbits.RC6 /* PORT for E */
#define TRIS_E DDRCbits.RC6 /* TRIS for E */
#endif

```

Es besteht also die Möglichkeit die verwendeten Ports für die Bibliothek anzupassen – man betrachte im Auszug die fett gedruckten Zeilen. Allerdings haben wir bis zu diesem Zeitpunkt keine Beschreibung der verwendeten Ports für die auf dem Board verwendete Peripherie gefunden. Es besteht also im Moment keine Möglichkeit die Bibliothek anzupassen was, zumindest für die „Software Peripherie Bibliothek“, bedeutet, dass wir sie nicht verwenden können bevor wir die Ports herausgefunden haben.

Es sollte noch bemerkt werden, dass die bei dem oben abgedruckten Auszug benutzen Ports nicht die auf dem PICDEM.net Board verwendet Ports sind. Das ist ein Original Auszug nach der Installation des C18 Compilers.

Die genaue Beschreibung für das LCD-Display folgt weiter unten.

Da nun festgestellt wurde dass die Bibliothek für das Ansprechen des LCD-Displays, da die genaue Portbelegung nicht bekannt ist, nicht ausreicht, ist die letzte Möglichkeit das „reverse Engineering“ durch den mitgelieferten MCHPStack von Microchip

3.2 MCHPStack von Microchip – Reverse Engineering

Durch das eben beschriebene Problem ist hier ein kleines Kapitel über den MCHPStack entstanden. Es soll deutlich gemacht werden wie wir bei Untersuchungen nach Port- und Pinbelegungen vorgegangen sind.

3.2.1 MCHPStack als einzige Quelle bei externe Peripherie

Der MCHPStack ist die einzige Quelle mit Angaben bezüglich Port- und Pinbelegung. Wenn die Demo-Firmware auf dem Microcontroller programmiert ist und man den Reset-Button drückt so kann man die aktuelle Version des Stack sowie die aktuelle IP-Adresse von dem LCD-Display ablesen. Daher hat der MCHPStack das LCD Display in seinem Code angesprochen.

3.2.2 Durchsuchen des Quellcodes von MCHPStack am Beispiel der Port- und Pinbelegung des LCD-Display zur Ansteuerung

Wir haben gesehen dass nach dem Start der Anwendung auf dem LCD-Display Ausgaben gemacht werden. Also muss das LCD-Display irgendwie am Anfang der main()-Methode oder in einer Init-Funktion angesprochen werden.

Die main()-Methode ist in der Quellcode Datei websrvr.c. Hier wird die ganze Anwendung initialisiert und der Main-Loop wird hier gehalten.

Wenn man nun in die Main Methode schaut kann man folgenden Ausschnitt erkennen:

```
/*
 * Main entry point.
 */
void main(void)
{
    TBLPTRU = 0x00;

    /*
     * Initialize any application specific hardware.
     */
    InitializeBoard();

    /*
     * Initialize all stack related components.
     * Following steps must be performed for all applications using
     * PICmicro TCP/IP Stack.
     */
    TickInit();
    .
    .
    .
}
```

Die fett gedruckten Zeilen führen zur Initialisierung der Hardware die für die Anwendung genutzt werden soll.

Ein Ausschnitt aus der Funktion InitializeBoard() zeigt nun folgendes:

```
static void InitializeBoard(void)
{
    /*
     * Setup for PORTA.RA0 as analog input while rests
     * as digital i/o lines.
     */
    ADCON1 = 0b10001110;      // RA0 as analog input, Right justified
    TRISA  = 0x03;

    /*
     * LCD is enabled using RA5.
     */
    PORTA_RA5 = 0;           // Disable LCD.

    /*
     * Turn off the LED's.
     */
    LATA2 = 1;
    LATA3 = 1;

    /*
     * External data EEPROM needs pull-ups, so enable internal
     * pull-ups.
     */
    INTCON2_RBPU = 0;

    XLCDInit();
    XLCDGoto(0, 0);
    XLCDPutROMString(StartupMsg);

    TXSTA = 0b00100000;      // Low BRG speed
    RCSTA = 0b10010000;
    SPBRG = SPBRG_VAL;

    T0CON = 0;
    INTCON GIEH = 1;
    INTCON GIEL = 1;
}
```

```
}

```

Hier ist erkennbar (fett gedruckt), dass das LCD-Display initialisiert wird – der Funktionsname ist in dem Fall selbsterklärend.

Weiter kann man auch noch andere Funktionen erkennen die auch selbsterklärend sind.

Da nun diese Funktionen einfach aufgerufen worden sind müssen sie bekannt sein. Wenn man nun an den Anfang der websrvr.c Datei geht und danach sucht wird man den entsprechenden Header Eintrag finden.

```
#include "xlcd.h"
.
.
.
ROM char StartupMsg[] = "MCHPStack v2.11";

```

Ein wenig weiter unten findet man auch noch die statische Nachricht die einem auf das LCD-Display bei dem Start der Anwendung geschrieben wird.

Natürlich ist der nächste Schritt die xlcd.h Datei gewesen und wenn man sich diese Anschaut wird man unter anderem folgenden Auszug finden:

```
/* DATA PORT defines the port to which the LCD data lines are connected */
#define DATA_PORT      PORTD
#define TRIS_DATA_PORT TRISD
.
.
.
/* CTRL PORT defines the port where the control lines are connected.
 * These are just samples, change to match your application.
 */

#define RW_PIN    PORTD_RD5  /* PORT for RW */
#define TRIS_RW   TRISD_RD5  /* TRIS for RW */
#define RS_PIN    PORTD_RD4  /* PORT for RS */
#define TRIS_RS   TRISD_RD4  /* TRIS for RS */
#define E_PIN     PORTA_RA5   /* PORT for E */
#define TRIS_E    TRISA_RA5   /* TRIS for E */

```

Die Information, die wir zu Begin gesucht haben ist hier abgebildet – die Port- und Pinbelegung für das LCD-Display.

Wenn wir nun den Auszug aus der C18 – Bibliothek für das LCD-Display und den Auszug aus dem MCHP-Stack miteinander vergleichen so kann man erkennen wo die Unterschiede sind.

MCHPStack	C18-Bibliothek
<pre>#define DATA_PORT PORTD #define TRIS_DATA_PORT TRISD #define RW_PIN PORTD #define TRIS_RW TRISD #define RS_PIN PORTD #define TRIS_RS TRISD #define E_PIN PORTA_ #define TRIS_E TRISA_ </pre>	<pre>#define DATA_PORT PORTB #define TRIS_DATA_PORT TRISB #define RW_PIN PORTBbits.RB6 #define TRIS_RW DDRBbits.RB6 #define RS_PIN PORTBbits.RB5 #define TRIS_RS DDRBbits.RB5 #define E_PIN PORTBbits.RB4 #define TRIS_E DDRBbits.RB4 </pre>

Der Datenport für das LCD-Display liegt bei dem PICDEM.net Board an Port D während es bei vielen anderen Demo-Boards und deshalb wahrscheinlich auch in der Bibliothek an Port B liegt.

Leider können wir jetzt immer noch nicht den Header der Bibliothek umschreiben, denn zu dem XLCD-Display gehören noch weitere spezifische Daten die an dieser Stelle immer noch nicht zur Verfügung stehen wie z.B. die Anzahl der genauen Zeichen die pro Zeile geschrieben werden können, die Pixelbreite eines Zeichens usw..

Es wird leider noch nicht einmal gesagt welcher Spezielle Typ des LCD-Display hier verwendet wird – lediglich der Hersteller (Hitachi) war versteckt in der mitgelieferten Dokumentation zu finden. Daher muss noch sehr viel tiefer im MCHPStack „herumgewühlt“ werden bis alle genauen Daten vorhanden sind. Dieses Wissen ist zur Initialisierung des LCD-Controllers notwendig.

3.2.3 Ergebnis des „reverse Engineering“

Als Fazit für diesen speziellen Fall kann man folgendes sagen: Es hat eine lange Zeit an Internet Recherchen gebraucht bis wir ein Datenblatt genau für diesen Controller des LCD-Display gefunden haben (ist auf der CD verfügbar). Zudem musste noch die spezifischen Daten aus den Quellcode Dateien des MCHPStacks gesucht werden um am Ende den genauen Typ des LCD-Displays feststellen zu können. Die ganz genauen Ergebnisse dieser und anderer Untersuchungen kann man im weiteren Verlauf dieses Kapitels sehen.

Es konnte hier deutlich gemacht werden was die einzige Möglichkeit war an die genauen Port- und Pinbelegungen heranzukommen.

Erkennbar war auch, dass alle Ports und Pins in der compiler.h Datei durch ein #define ersetzt wurden. Dadurch haben wir später auch die genauen Pins für die zwei Dioden gefunden welche wir ansteuern wollten.

Für unsere Anwendung haben wir die verwendeten Bibliotheks-Funktionen (z.B.: xlcd.h und xlcd.c) von dem MCHPStack übernommen und für unsere eigenen Zwecke abgewandelt. Dies war wesentlich einfacher als die gegebenen C18-Bibliotheks-Funktionen um zu schreiben.

So wurde die von den C18 Compiler mitgelieferten Bibliotheken die Gruppe „Software Peripherie Bibliothek“ durch eigene bzw. übernommene und abgewandelte Bibliotheks-Funktionen des MCHPStacks ersetzt.

3.3 **Microcontroller PIC18F452**

Das PICDEM.net Demo Board wird von Haus aus mit dem PIC18F452 Microcontroller bestückt. Dieser Microcontroller ist weit verbreitet und es gibt viele Anwendungen und Beispiele die mit diesem arbeiten. Das wohl mit am weitesten verbreitete Demo Board aus dem Hause Microchip mit dem Microcontroller PIC18F452 ist das PICDEM 2 Plus Demo Board.

3.3.1 RISC (Reduced Instruction Set Computer) CPU

- Linearer Program Speicher – 32 KByte Adressierbar
- Linearer Daten Speicher – 1.5 KByte Adressierbar

- Speicher Übersicht:

On-Chip Program Memory		On-Chip RAM (bytes)	Data EEPROM (bytes)
FLASH (bytes)	# Single Word Instructions		
32 K	16384	1536	256

- 16 Bit große Instruktionen, 8 Bit Datenleitung
- Prioritätslevel für Interrupts

3.3.2 Peripherie

- 3 externe Interrupt Pins
- Timer0 Modul: 8-Bit oder 16-Bit Timer/Counter mit 8-Bit Rücksetzwert
- Timer1 Modul: 16-Bit Timer/Counter
- Timer2 Modul: 8-Bit Timer/Counter mit 8-Bit periodischem Register (Zeitbasis für PWM)
- Timer3 Modul: 16-Bit Timer/Counter
- Zwei Capture/Compare/PWM Module
- Adressierbarer USART Modul
- I²C Bus
- 8 x 8 Hardware Multiplizierer
- 5 Digitale I/O – Ports
- Master Synchronous Serial Port

3.3.3 Analoge Eigenschaften

- 10-Bit Analog-Digital Wandler mit:
 - Fast sampling rate
 - Umwandlung während SLEEP Mode
 - Linearität < 1 LSB
- Programmierbare Low-Voltage (Zu wenig Spannung) Erkennung

3.3.4 Spezielle Microcontroller Eigenschaften

- Power-On Reset
- Power-Up Timer
- Watchdog Timer
- Programmierbarer Code Schutz (Code Protection)
- Power saving SLEEP Mode
- Auswählbare Oszillator Optionen
- In-Circuit Programmierung
- In-Circuit Debugging

3.3.5 Pin Diagram

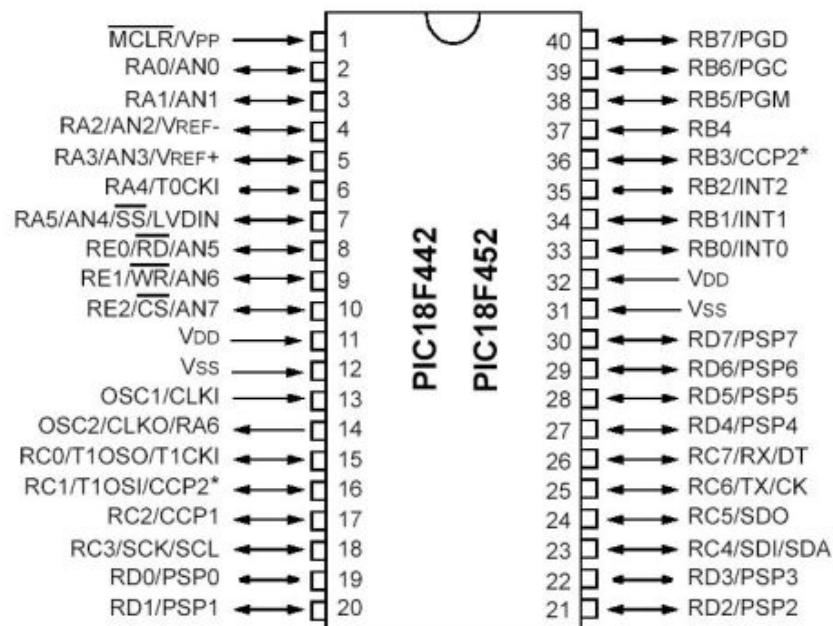


Bild 11 PIC18F452 Pin Belegung

3.3.6 Gerät Eigenschaften Überblick

Eigenschaft	PIC18F452
Arbeitsfrequenz	DC – 40 MHz
Programm Speicher (Bytes)	32 K
Programm Speicher (Instruktionen)	16384
Daten Speicher (Bytes)	1536
Daten EEPROM Speicher (Bytes)	256
Interrupt Quellen	18
I/O Ports	Ports A, B, C, D, E
Timer	4
Capture/Compare/PWM Module	2
Serielle Kommunikation	MSSP, Adressierbarer USART
Parallele Kommunikation	PSP
10-Bit Analog-Digital-Wandler	8 Eingangs Kanäle
RESET und Verzögerungen	POR, BOR, RESET Instruktion, Stack Full, Stack Underflow, (PWRT, OST)
Programmierbarer LOW-Voltage Detect	Ja
Programmierbarer Brown-Out Reset	Ja
Instruction Set	75 Instruktionen
Packet	40-Pin Dip

3.4 Externe verfügbare Peripherie

Der Vollständigkeit wegen wird an dieser Stelle auch alle externe verfügbare und ansteuerbare Peripherie aufgezählt wobei die exakte Beschreibung der Ansteuerung und Handhabung der Peripherie im Kapitel 3 zu finden ist.

Peripherie	Zusatzinformationen
Speicherbaustein 256Kbit	ansteuerbar über den I ² C Bus
Hitachi LCD-Display	2 Zeilen jeweils 16 Zeichen
Ethernet Controller	Realtek RTL8019AS Controller
User-defined LEDs	An zwei digitalen Ausgängen des Controllers
User-defined Push-Button	An einen digitalen Pin des Controllers angeschlossen
RS 232 Schnittstelle	Serieller Port
Prototyp Bereich	An den Ausgängen der Ports angeschlossen (siehe Kapitel 1)

3.5 Generelle Erkenntnis / Fazit

Dieses Kapitel beschreibt eine wichtige Phase des Projekts was sehr viele Erkenntnisse über das Programmieren des PIC18F452, der Programmiersprache selbst und der Bestückung des PICDEM.net Demo Board preisgibt. Alle Informationen die hier zusammen getragen wurden kann man durch eigene Recherche in den mitgelieferten pdf-Dokumentationen und des MCHPStacks herausfinden was allerdings auch sehr mühsam ist.

Eines der Hauptanliegen, weshalb es zu dieser Ausführlichen Untersuchung auch erst gekommen ist, war das LCD-Display. Da dieses auf alle Fälle genutzt werden sollte und es keine spezielle Dokumentation bezüglich dieses Displays gibt, ist es zwangsweise zur der eben Beschriebenen Entwicklung gekommen. Das nun eigentliche und vielleicht auch interessantere Ansprechen und die Ergebnisse dieser Untersuchung werden im nächsten Kapitel, dem Ansprechen der Peripherie und Hardware, beschrieben.

4 Ansprechen der Peripherie und Hardware in C

Im Kapitel 2 wurde beschrieben welche Hardware auf dem PICDEM.net Demo Board uns zur Verfügung steht.

Da nun auch die grundlegenden Informationen über den Microcontroller, die Programmiersprache mit Compiler C18 und der Peripherie des PICDEM.net Board zur Verfügung stehen soll Erläutert werden wie die einzelne Peripherie angesteuert werden kann.

Zudem sollte noch einmal erwähnt werden, dass Teile aus dem Source-Code des MCHPStacks von Microchip übernommen und für eigene Zwecke abgeändert wurden.

4.1 Allgemeine Definitionen - defines.h

In unserem Source-Code inkludieren wir immer die Datei defines.h da diese alle globalen Definitionen wie z.B. selbst definierte Datentypen oder die Clock-Frequenz.

Auszug aus defines.h

```
// Datentypen festlegen
typedef unsigned char BYTE;      // 8-bit
typedef unsigned short int WORD; // 16-bit

#define CLOCK_FREQ      (20000000)    // Hz

typedef union _WORD_VAL
{
    WORD Val;
    BYTE v[2];
} WORD_VAL;
```

4.2 User-LEDs

Eines der bekanntesten Ansteuerungen die wohl überall verwendet werden sind LEDs. Daher soll an diesem einfachen Beispiel deutlich gemacht werden wie man durch die spezifische Mikroprozessor Bibliothek Ports konfiguriert und auf diese zugegriffen wird.



Auf dem PICDEM.net Demo Board gibt es 2 LEDs die von einer Anwendung aus genutzt werden können. Diese beiden LEDs können zudem noch zudem manuell von einem Jumper, der oberhalb der rechten LED zu sehen ist, aktiviert oder deaktiviert werden. Wenn der Jumper gezogen ist können diese USER-LEDs nicht mehr angesprochen werden.

Das Bild wurde von unserer ersten Anwendung gemacht bei dem beide LEDs einfach abwechselnd blinken. Man kann erkennen, dass die linke LED gelb leuchtet.

Durch unsere Untersuchungen haben wir folgende Portpins für die LEDs ausfindig gemacht:

LED 1 (links)	PORT A	PIN 2
LED 2 (rechtes)	PORT A	PIN 3

Aber zunächst muss der Port A noch als Output konfiguriert werden. In der Dokumentation des Microcontrollers werden in Kapitel 9, I/O Ports, alle vorhandenen Ports beschrieben.

Für den Port A gilt als default Einstellung bei einem Power-Reset, dass die Pins RA5 und RA3:RA0 (liest sich: RA3 bis RA0) als analoge Inputs konfiguriert sind. Die Pins RA6 und RA4 sind als digitale Inputs konfiguriert.

Da wir allerdings die Pins RA3 und RA2 als digitale Output benutzen möchten müssen wir das „Data Direction Register“ TRISA auf 0 setzen.

Die Portpins, an denen die Dioden angeschlossen sind, können nun mit folgender Struktur angesprochen werden:

```
DDRAbits.RA2 = 0; // an
DDRAbits.RA3 = 1; // aus
```

Die Dioden besitzen eine negative Logik was bedeutet, dass sie bei einer 0 an- bzw. einer 1 ausgeschaltet werden.

Da der gesamte Code für dieses Programm nur sehr gering ist wird dieser hier aufgelistet.

```
#include <p18cxxx.h>
#include <delays.h>

void main (void)
{
    /*
     * PORT A als Output konfigurieren
     * Default ist: RA5 und RA3:RA0 digital input und read
     */
    TRISA = 0; // Output

    /*
     * Die zwei LEDs hängen an dem Port A -> Bit 2 & 3
     * LEDs initialisieren -> inverse Logik -> 0xFF
     * 1. LED -> an
     * 2. LED -> aus
     */
    DDRAbits.RA2 = 0; // an
    DDRAbits.RA3 = 1; // aus

    while(1)
    {
        /* Anliegenden Wert an LEDs invertieren */
        DDRAbits.RA2 ^= 1;
        DDRAbits.RA3 ^= 1;

        /* Wartet ein vielfaches von 10000 ProzessorZyklen */
        Delay10KTCYx(1000);
    }
}
```

Im Code erkennt man, dass erst der Port A als Output konfiguriert wird, dann werden die Dioden in ihren Start Zustand gesetzt und später der Wert der anliegt einfach invertiert.

Man kann noch eine Bibliotheks-Funktion erkennen (`Delay10KTCYx(1000)`) die den Loop soweit verzögert, dass das blinken mit unserem trägen Auge zu erkennen ist.

Die beschriebenen Stellen sind hier nicht fett gedruckt, da der abgedruckte Code ausreichend und selbsterklärend kommentiert ist.

4.3 LCD Display

Eine der größeren Herausforderungen in diesem Projekt war mit Sicherheit das verfügbare LCD-Display. Außer der allgemeinen Angabe, dass das LCD-Display von Hitachi ist wurde keinerlei Angabe diesbezüglich gemacht. Leider waren die Angaben der Portbelegung für den Controller des LCD-Displays nicht verfügbar. Aber genau diese Information ist Notwendig wenn man den Controller ansprechen möchte. Zudem war nicht klar welche Control-Wörter notwendig sind um das LCD-Display zu konfigurieren oder anzusprechen. Daher war die erste Aufgabe herauszufinden welchen genauen Typ von LCD-Display von Hitachi wir verwenden.

4.3.1 Eigenschaften

Wenn man sich nun mit LCD-Displays beschäftigt kann man verschiedene Merkmale feststellen wie diese aufgebaut sind:

- x*y Pixel Zeichendarstellung
- Anzahl der verfügbaren Zeilen
- n Zeichen pro Zeile
- n-Bit Interface

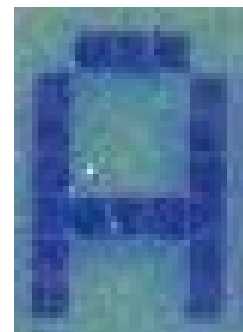
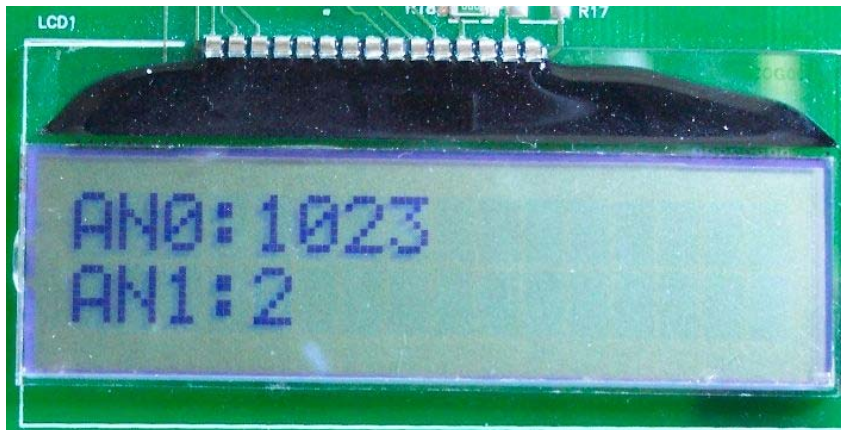


Bild 12 - Hitachi HD44780 LCD-Display

Nach Untersuchungen an unserem LCD-Display haben sich nun folgende Merkmale feststellen lassen:

- 5x8 Pixel Zeichendarstellung (rechtes Bild – Zeichen A)
- 2 Zeilen
- 16 Zeichen pro Zeile
- 4 Bit Interface

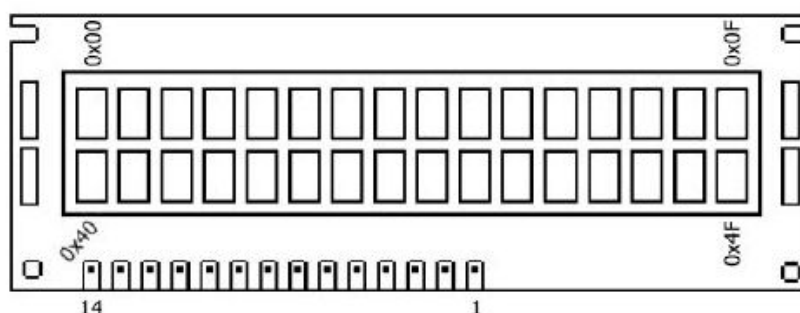


Bild 13 - Hitachi LCD-Display - 2 Zeilen, 16 Zeichen

4.3.2 LCD-Controller

An welchem Port nun das LCD-Display genau angeschlossen ist konnte wieder nur Anhand des mitgelieferten MCHPStack herausgefunden werden. Folgender Auszug aus dem xlcd-Header File zeigt die Port- und Pinbelegung.

Auszug aus xlcd.h

```
/* DATA_PORT defines the port to which the LCD data lines are connected */
#define DATA_PORT PORTD
#define TRIS_DATA_PORT TRISD

#define RW_PIN PORTDbits.RD5 /* PORT for RW */
#define TRIS_RW TRISDbits.TRISD5 /* TRIS for RW */
#define RS_PIN PORTDbits.RD4 /* PORT for RS */
#define TRIS_RS TRISDbits.TRISD4 /* TRIS for RS */
#define E_PIN PORTAbits.RA5 /* PORT for E */
#define TRIS_E TRISAbits.TRISA5 /* TRIS for E */
```

Die Anzahl der Zeichen und Zeilen auf dem Display konnte noch herausgefunden werden allerdings dass Wissen, dass es sich hier um ein 4-Bit Interface handelt wurde erst an dieser Stelle erweitert.

Somit war nun auch klar wie das LCD-Display konfiguriert werden kann. Es gibt dafür natürlich noch die Bibliotheksfunktion welche man auch der Einfachheit halber nutzen sollte. Dennoch wird hier noch darauf Eingegangen wie man die genauen Datenwörter sich Anhand des HITACHI LCD-Display Data-Sheets zusammenstellt.

Das genaue Beispiel beinhaltet die Konfiguration des genauen LCD-Display Typs. Es soll auf 4-Bit Interface, 5x8 Pixel Zeichengröße und 2 Zeilen konfiguriert werden.

Folgende Datenworttabelle zeigt alle Möglichkeiten der Datenwörterkombinationen.

Instruction	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Description	Clock-Cycles	
NOP	0	0	0	0	0	0	0	0	0	0	No Operation	0	
Clear Display	0	0	0	0	0	0	0	0	0	1	Clear display & set address counter to zero	165	
Cursor Home	0	0	0	0	0	0	0	0	1	x	Set adress counter to zero, return shifted display to original position. DD RAM contents remains unchanged.	3	
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Set cursor move direction (I/D) and specify automatic display shift (S).	3	
Display Control	0	0	0	0	0	0	1	D	C	B	Turn display (D), cursor on/off (C), and cursor blinking (B).	3	
Cursor / Display shift	0	0	0	0	0	1	S/C	R/L	x	x	Shift display or move cursor (S/C) and specify direction (R/L).	3	
Function Set	0	0	0	0	1	DL	N	F	x	x	Set interface data width (DL), number of display lines (N) and character font (F).	3	
Set CGRAM Address	0	0	0	1	CGRAM Address						Set CGRAM address. CGRAM data is sent afterwards.	3	
Set DDRAM Address	0	0	1	DDRAM Address						Set DDRAM address. DDRAM data is sent afterwards.	3		
Busy Flag & Address	0	1	BF	Address Counter						Read busy flag (BF) and address counter	0		
Write Data	1	0	Data						Write data into DDRAM or CGRAM	3			
Read Data	1	1	Data						Read data from DDRAM or CGRAM	3			
x : Don't care	I/D	1 0	Increment Decrement						R/L	1 0	Shift to the right Shift to the left		
	S	1 0	Automatic display shift						DL	1 0	8 bit interface 4 bit interface		
	D	1 0	Display ON Display OFF						N	1 0	2 lines 1 line		
	C	1 0	Cursor ON Cursor OFF						F	1 0	5x10 dots 5x7 dots		
	B	1 0	Cursor blinking						DDRAM : Display Data RAM CGRAM : Character Generator RAM				
	S/C	1 0	Display shift Cursor move										

Bild 14 - Datenworttabelle für HITACHI LCD-Display

Für die Konfiguration des LCD-Displays ist folgende Zeile wichtig:

Function Set	0	0	0	0	1	DL	N	F	x	x	Set interface data width (DL), number of display lines (N) and character font (F).	3
--------------	---	---	---	---	---	----	---	---	---	---	--	---

Es stellt sich also folgendes Datenwort heraus:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	DL	N	F	x	x

Unterhalb der Tabelle findet man nun noch folgende Auswahlmöglichkeiten für unser Datenwort:

Bezeichnung	Bit	Bedeutung
x	0/1	Don't care
DL	1 0	8 Bit Interface 4 Bit Interface
N	1 0	2 Zeilen 1 Zeile
F	1 0	5x10 Dots 4x7 Dots

Die entsprechenden Bits, die für unser LCD-Display gelten, sind hier fett markiert. Somit ergibt sich nun folgendes Datenwort welches, unter anderem, bei dem Initialisieren des LCD-Displays an den LCD-Controller gesendet werden muss.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	0	1	0	0	0

Wenn nun das Datenwort bekannt ist muss dieses noch an den LCD-Controller gesendet werden. Dazu gibt es nun folgende Regelung zur Erkennung einer Übertragung.

Bus Timing Characteristics

($T_a = -20$ to $+75^\circ\text{C}$)

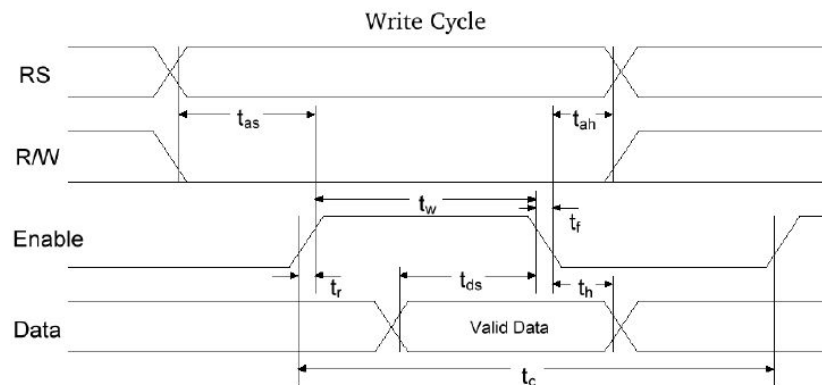


Bild 15 - LCD-Controller - Zeitcharakteristik bei Datenwortübertragung

Ohne auf die genauen Zeiten einzugehen (diese können im LCD-Display Data Sheet nachgelesen werden), kann man erkennen, dass Zeiten eingehalten werden müssen sobald ein Datenwort an den LCD-Controller gesendet werden soll.

Da es sich zusätzlich noch um ein 8-Bit Datenwort aber nur um ein 4-Bit Interface handelt muss das Datenwort in zwei Schritten gesendet werden.

Zeitliche Abfolge der Befehle laut Zeitcharakteristik:

1. RS auf 0
2. R/W auf 0
3. Adressierung vorbereiten – Zeit t_{as}
4. Enable-Bit auf 1 – für die Zeit t_w
5. Während das Enable-Bit auf 1 ist soll in der Zeit t_{ds} die Daten vorbereitet werden
6. Datenübertragung sobald Enable-Bit wieder auf 0

Natürlich muss der Port über den die Daten gesendet werden vorher als Output geschaltet werden.

Da bei jedem Kommando für das LCD-Display diese Prozedur durchgegangen werden muss, lohnt es sich hierfür eine Funktion zu schreiben. In der verfügbaren Bibliothek heißt diese dann `void XLCDCommand(unsigned char cmd)`.

Die gesamte hier durchgeführte Vorgehensweise kann man folgendermaßen in der XLCD-Bibliothek finden.

Auszug `xlcd.h`

```
/*
 * Set your LCD type.
```

```
*/
#define XCLD TYPE          (FOUR BIT & LINES 5X8)
.
.
.
/* Function Set defines */
#define FOUR BIT    0b00101111 /* 4-bit Interface */
#define LINES_5X8  0b00111000 /* 5x8 characters, multiple line */
```

Die gesamte Prozedur des Initialisierens findet sich in der Funktion `void XLCDInit(void)` aus der nun folgender Auszug kommt:

Auszug xlcd.c

```
void XLCDInit(void)
{
    .
    .
    XLCDCommand(XCLD_TYPE);    // Function set cmd
    .
    .
}
```

Da das Übertragen von Datenwörtern genauer betrachtet wurde, wir an dieser Stelle auch ein Auszug der Funktion `void XLCDCommand(unsigned char cmd)` gelistet.

```
void XLCDCommand(unsigned char cmd)
{
    while(XLCDIsBusy());    // Warten, falls LCD noch beschäftigt ist

    TRIS_RW      = 0;    // Alle Controlle Signale als Output konfigurieren
    TRIS_RS      = 0;

    // Lower nibble interface
    TRIS_DATA_PORT &= 0xf0; // Die unteren 4-Bits als Output konfigurieren
    DATA_PORT &= 0xf0;    // Den oberen 4-Bits auf 1, die unteren auf 0 setzen
    DATA_PORT |= (cmd>>4); // Die oberen 4-Bits des Datenwort nach rechts schieben

    RW_PIN = 0;    // Controlle Signale für Kommando setzen
    RS_PIN = 0;
    XLCDDelay500ns();
    E_PIN = 1;    // Enable Signal auf 1
    XLCDDelay500ns();
    E_PIN = 0;    // Enable Signal auf 0

    // Daten wurden direkt nach den Enable Signal gesendet

    // Lower nibble interface
    DATA_PORT &= 0xf0;    // Data Port wieder vorbereiten
    DATA_PORT |= cmd&0xf; // nur die unteren 4-Bits anlegen

    XLCDDelay500ns();
    E_PIN = 1;    // Enable Signal auf 1
    XLCDDelay500ns();
    E_PIN = 0;    // Enable Signal auf 0

    // Daten wurden direkt nach den Enable Signal gesendet

    // Data Pins/Port wieder als Input konfigurieren

    TRIS_DATA_PORT |= 0x0f;

    return;
}
```

An dem letzten Auszug kann man erkennen, wie das Datenwort in 2 Teile zerlegt wurde und dass die jeweiligen Teile immer schon vor dem setzen des Enable-Bits an

den Data-Port gelegt wurden. Somit konnte das anliegende Datenwort gleich nach dem Rücksetzen des Enable-Bits auf 0 gesendet werden.

Das HITACHI HD44780 LCD-Display Data-Sheet ist als pdf im Unterordner /pdf/ zu finden.

4.3.3 Bibliotheksfunktionen

Da es nun viele Prozeduren und Kommandos gibt die sich ständig wiederholen und da es viel zu kompliziert ist diese ständig manuell auszuführen ist eine solche Bibliothek nur sinnvoll.

Unsere verwendete Bibliothek wurde von dem MCHPStack übernommen und angepasst.

Name der Bibliothek	xlcd – HITACHI LCD-Display
Header File	xlcd.h
Source File	xlcd.c
Funktionen	
<code>void XLCDInit(void)</code>	
Initialisierung des HITACHI LCD-Display. • 4-Bit Interface • 5x8 Pixel • 2 Zeilen • Cursor off – nicht sichtbar • Blink off – blinkt nicht • Display wird gelöscht	
<code>void XLCDCommand(unsigned char cmd)</code>	
Die übergebenen Kommandos werden in 4-Bit Teilen an das LCD-Display als Kommando gesendet. Ein genaues Beispiel dazu findet sich im Kapitel vorher bei der Initialisierung des LCD-Display. Diese Funktion wird hauptsächlich von anderen Bibliotheksfunktionen genutzt.	
<code>void XLCDPut(char data)</code>	
Schreibt ein einzelnes Zeichen auf das Display an die Stelle an die der Cursor steht.	
<code>void XLCDPutString(char *string)</code>	
Schreibt eine Zeichenkette/String auf das Display beginnend an der Stelle an der der Cursor steht.	
<code>void XLCDPutROMString(rom char *string)</code>	
Schreibt eine Zeichenkette die im ROM liegt auf das Display beginnend an der Stelle an der der Cursor steht.	
<code>char XLCDIsBusy(void)</code>	
Es wird überprüft ob das LCD-Display noch beschäftigt ist. Ist dies der Fall wird ein Wert ungleich 0 zurückgegeben. Wenn es bereit für neue Kommandos oder nicht mehr beschäftigt ist wird eine 0 zurückgegeben.	
<code>void XLCDGoto(int row, int col)</code>	
Zur Steuerung des Cursors kann man die Zeile und Spalte angeben wohin der	

Cursor gestellt werden soll.

```
void XLCDClear(void)
```

Löscht das LCD-Display und setzt den Cursor an die Stelle 0,0.

Es gibt noch weitere Funktionen, die aber nur sehr selten und in unserem Projekt überhaupt nicht genutzt wurden. Diese Funktionen sind dadurch hier auch nicht aufgelistet können aber in dem Header File `xlcd.h` nachgelesen werden.

4.3.4 Ansteuerung in C

Hier eine einfache Ansteuerung in C, welche bei dem Tutorial „Analog-Digital-Wandler auf dem LCD-Display ausgeben“ weiter unten gebraucht wird.

```
#include "xlcd.h"

void main(void)
{
    // LCD Display initialisieren
    XLCDInit();
    XLCDClear();

    // Warten bis LCD-Display nicht mehr beschäftigt
    while(XLCDIsBusy());

    // In die erste Zeile schreiben
    XLCDGoto(0,0);
    XLCDPutString("Hello");

    // Warten bis LCD-Display nicht mehr beschäftigt
    while(XLCDIsBusy());

    // In die zweite Zeile schreiben
    XLCDGoto(1,0);
    XLCDPutString("World");
}
```

4.4 **Serielle Schnittstelle**

4.4.1 Eigenschaften

Die serielle Schnittstelle wird durch den internen USART (Universal Synchronous Asynchronous Receive Transmit) realisiert. Wie der Name schon sagt hat diese Schnittstelle einige erweiterte Eigenschaften gegenüber dem Standard UART.

Im asynchronen Modus verhält sich der USART wie ein UART, und ermöglicht somit die Kommunikation mit verschiedener Peripherie, wie Modems oder CRTs.

Im synchronen Modus (möglich als Master oder als Slave) kann er mit Peripherie, wie A/D oder D/A ICs oder seriellen EEPROMs kommunizieren.

Wir wollen hier hauptsächlich auf die Konfiguration und den Betrieb als UART eingehen.

4.4.2 Register

Der USART verfügt über 5 Register, die zum Betrieb benötigt werden:

- Transmit Status and Control Register(TXSTA)
- Receive Status and Control Register(RCSTA)
- Baud Rate Generator Register(SPBRG)
- USART Transmit Register(TXREG)
- USART Receive Register(RCREG)

Bit 7	6	5	4	3	2	1	Bit 0
CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D

Da die Tx-Leitung mit dem Pin6 und die Rc-Leitungen mit dem Pin7 des digitalen I/O-Ports C gemultiplext sind, muss Bit 6 des TRISC – Registers 0 (Output) und Bit 7 1 (Input) gesetzt sein. Um das interruptgesteuerte Senden und Empfangen zu ermöglichen müssen noch diverse Register des Interruptcontrollers konfiguriert werden. Aber darauf wollen wir später eingehen.

Register TXSTA:

Stelle	Bezeichnung	Bedeutung							
7	CSRC	Clock Source Select Bit	RW – Default 0						
		Im Asynchronen Modus: Keine Auswirkungen							
		Im Synchronen Modus:							
		<table><tr><th>Bit</th><th>Auswirkung</th></tr><tr><td>1</td><td>Master Modus (Takt wird vom internen Baud Rate Generator generiert)</td></tr><tr><td>0</td><td>Slave Modus (Takt kommt aus externer Quelle)</td></tr></table>	Bit	Auswirkung	1	Master Modus (Takt wird vom internen Baud Rate Generator generiert)	0	Slave Modus (Takt kommt aus externer Quelle)	
Bit	Auswirkung								
1	Master Modus (Takt wird vom internen Baud Rate Generator generiert)								
0	Slave Modus (Takt kommt aus externer Quelle)								
6	TX9	9-bit Transmit Enable Bit	RW – Default 0						
		<table><tr><th>Bit</th><th>Auswirkung</th></tr><tr><td>1</td><td>Es werden immer 9 Bit gesendet</td></tr><tr><td>0</td><td>Es werden immer 8 Bit gesendet</td></tr></table>	Bit	Auswirkung	1	Es werden immer 9 Bit gesendet	0	Es werden immer 8 Bit gesendet	
Bit	Auswirkung								
1	Es werden immer 9 Bit gesendet								
0	Es werden immer 8 Bit gesendet								
5	TXEN	Transmit Enable Bit	RW – Default 0						
		<table><tr><th>Bit</th><th>Auswirkung</th></tr><tr><td>1</td><td>Transmit wird eingeschaltet</td></tr><tr><td>0</td><td>Transmit wird ausgeschaltet</td></tr></table>	Bit	Auswirkung	1	Transmit wird eingeschaltet	0	Transmit wird ausgeschaltet	
Bit	Auswirkung								
1	Transmit wird eingeschaltet								
0	Transmit wird ausgeschaltet								
4	SYNC	USART Mode Select Bit	RW – Default 0						
		<table><tr><th>Bit</th><th>Auswirkung</th></tr><tr><td>1</td><td>Synchroner Modus</td></tr><tr><td>0</td><td>Asynchroner Modus</td></tr></table>	Bit	Auswirkung	1	Synchroner Modus	0	Asynchroner Modus	
Bit	Auswirkung								
1	Synchroner Modus								
0	Asynchroner Modus								
3	Unbenutzt	Wird als 0 gelesen							
2	BRGH	High Baud Rate Select Bit	RW – Default 0						
		Im Asynchronen Modus:							
		<table><tr><th>Bit</th><th>Auswirkung</th></tr><tr><td>1</td><td>Hohe Baud Rate</td></tr><tr><td>0</td><td>Niedrige Baud Rate</td></tr></table>	Bit	Auswirkung	1	Hohe Baud Rate	0	Niedrige Baud Rate	
Bit	Auswirkung								
1	Hohe Baud Rate								
0	Niedrige Baud Rate								

		Im Synchronen Modus: Keine Auswirkungen	
1	TRMT	Transmit Shift Reg Status Bit R – Default 1	
		Bit	Auswirkung
		1	TSR ist leer
		0	TSR ist voll
0	TX9D	Bit 9 der Transmit Daten RW – Default 0	

Stelle	Bezeichnung	Bedeutung	
7	SPEN	Serial Port Enable Bit RW – Default 0	
		Bit	Auswirkung
		1	Master Modus (Takt wird vom internen Baud Rate Generator generiert)
		0	Slave Modus (Takt kommt aus externer Quelle)
6	RX9	9-bit Receive Enable Bit RW – Default 0	
		Bit	Auswirkung
		1	Es werden immer 9 Bit empfangen
		0	Es werden immer 8 Bit empfangen
5	SREN	Single Receive Enable Bit RW – Default 0	
		Im Asynchrone Modus: Keine Auswirkungen	
		Im Synchronen Modus – Master:	
		Bit	Auswirkung
		Nach Empfang wird Bit zurückgesetzt	
4	CREN	Continuous Receive Enable Bit RW – Default 0	
		Im Asynchrone Modus:	
		Bit	Auswirkung
		1	Receiver wird eingeschaltet
		0	Receiver wird ausgeschaltet
Im Synchronen Modus:			
Bit	Auswirkung		
1	USART nimmt Pakete an bis CREN zurückgesetzt wird (überschreibt SREN)		
0	USART nimmt keine Pakete an		

3	ADDEN	Address Detect Enable Bit						RW – Default 0
2	FERR	Framing Error Bit						R – Default 0
Bit 7	6	5	4	3	2	1	Bit 0	
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	
		neu geprüft)						
		0	Kein Framing Error					
		Im Synchronen Modus: Keine Auswirkungen						
1	OERR	Overrun Error Bit						R – Default 0
		Bit	Auswirkung					
		1	Overrun Error (Wird durch zurücksetzen des CREN geloescht)					
		0	Kein Overrun Error					
0	RC9D	Bit 9 der Receive Daten						RW – Default 0

Register RCSTA:

Register SPBRG:

Dieses Register wird benutzt um die Baudrate zu bestimmen. Diese errechnet sich aus folgenden können aus folgenden Formeln berechnet werden:

Modus	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
Sync	$\text{Baud} = \text{Fosc}/(64(\text{SPBRG}+1))$	$\text{Baud} = \text{Fosc}/(16(\text{SPBRG}+1))$
Async	$\text{Baud} = \text{Fosc}/(4(\text{SPBRG}+1))$	N/A

Der PIC18F452 besitzt einen internen Oszillatortakt von 20MHz. Damit lassen sich folgende Tabellen aufstellen:

Synchron			Asynchron BRGH = 0			Asynchron BRGH = 1		
KBaud Soll	KBaud	SPBRG	KBaud Soll	KBaud	SPBRG	KBaud Soll	KBaud	SPBRG
0.3	N/A	-	0.3	N/A	-	0.3	N/A	-
1.2	N/A	-	1.2	N/A	-	1.2	N/A	-
2.4	N/A	-	2.4	2.40	129	2.4	N/A	-
9.6	N/A	-	9.6	9.47	32	9.6	9.62	129
19.2	N/A	-	19.2	19.53	15	19.2	19.23	64
76.8	76.92	129	76.8	78.13	3	76.8	78.13	15
96	96.15	103	96	104.17	2	96	96.15	12
300	303.03	32	300	312.5	0	300	312.50	3
500	500	19	500	N/A	-	500	416.67	2

Um eine möglichst kleine Abweichung von der gewünschten Baudrate zu erreichen, empfiehlt es sich BRGH auf High zu setzen.

Register TXREG:

Dies ist das Transmit Register. Dort werden die zu sendenden Daten hinein geladen.

Register RCREG:

Das Receive Register. Hier können empfangene Daten abgerufen werden.

4.4.3 Konfigurieren des USART in C

Um nun den USART als UART zu konfigurieren müssen wir die vorgestellten Register entsprechend initialisieren.

```
TXSTAbits.BRGH = 1; // Baudrate High Speed select
SPBRG = 64;      // High Speed + Async -> 19200 Baud
TXSTAbits.SYNC = 0; // Asynchroner Modus
RCSTAbits.SPEN = 1; // Serial Port Enabled
RCSTAbits.RX9 = 0; // 8-Bit Empfang
RCSTAbits.CREN = 1; // Enable Receiver
TXSTAbits.TXEN = 1; // Enable Transmitter
TRISCbits.TRISC6 = 0; // TxPin -> Output
TRISCbits.TRISC7 = 1; // RcPin -> Input
```

Damit sollte der USART Betriebsbereit sein. Wird nun ein Paket empfangen, wird das RCIF Bit im PIR1 Register (auf dieses Register wird im Kapitel Interrupts näher eingegangen) gesetzt. Nach dem Auslesen des Pakets aus dem RCREG Register wird das Bit automatisch gelöscht.

```
if(PIR1bits.RCIF == 1) // Paket wurde empfangen
    data = RCREG; // Paket ausgelesen, Bit wurde zurückgesetzt
```

Soll nun ein Paket gesendet werden, muss zuerst geprüft werden, ob das Transmit Shift Register leer ist. Dies macht man indem man nachschaut, ob das TRMT Bit im TXSTA Register gesetzt ist. Ist das der Fall kann man das Paket direkt in das TXREG Register laden, was des Sendevorgang sofort einleitet.

```
if(TXSTAbits.TRMT == 1) // Transmit Shift Register ist leer
    TXREG = data; // Daten werden sofort losgeschickt
```

4.5 AD-Wandler

4.5.1 Eigenschaften

Das Analog-Digital-Wandler Modul hat bei dem PIC18F4x2 8 Eingänge und einen Auflösungsbereich von 10 Bit für ein anliegendes Analoges Signal.

Es gibt folgende vier Register für das A/D Modul:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register 0 (ADCON0)
- A/D Control Register 1 (ADCON1)

Das ADCON0 Register kontrolliert die Operationen des A/D Moduls. Das ADCON1 Register konfiguriert die Funktionen der Portpins.

4.5.2 Register Konfiguration

ADCON0 Register:

Bit 7	6	5	4	3	2	1	Bit 0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/-DONE	-	ADON

Konfiguration der einzelnen Bits:

Stelle	Bezeichnung	Bedeutung																											
7-6	ADCS1:ADCS0	A/D Taktgeber für Umwandlungszeit-Auswahl Bits <table><tr><th>ADCON1 ADSC2</th><th>ADCON0 ADSC1:ADSC0</th><th>Taktgeberumwandlung</th></tr><tr><td>0</td><td>00</td><td>Fosc/2</td></tr><tr><td>0</td><td>01</td><td>Fosc/8</td></tr><tr><td>0</td><td>10</td><td>Fosc/32</td></tr><tr><td>0</td><td>11</td><td>FRC (Abhängig vom internen A/D RC Oszillator)</td></tr><tr><td>1</td><td>00</td><td>Fosc/4</td></tr><tr><td>1</td><td>01</td><td>Fosc/16</td></tr><tr><td>1</td><td>10</td><td>Fosc/64</td></tr><tr><td>1</td><td>11</td><td>FRC (Abhängig vom internen A/D RC Oszillator)</td></tr></table>	ADCON1 ADSC2	ADCON0 ADSC1:ADSC0	Taktgeberumwandlung	0	00	Fosc/2	0	01	Fosc/8	0	10	Fosc/32	0	11	FRC (Abhängig vom internen A/D RC Oszillator)	1	00	Fosc/4	1	01	Fosc/16	1	10	Fosc/64	1	11	FRC (Abhängig vom internen A/D RC Oszillator)
ADCON1 ADSC2	ADCON0 ADSC1:ADSC0	Taktgeberumwandlung																											
0	00	Fosc/2																											
0	01	Fosc/8																											
0	10	Fosc/32																											
0	11	FRC (Abhängig vom internen A/D RC Oszillator)																											
1	00	Fosc/4																											
1	01	Fosc/16																											
1	10	Fosc/64																											
1	11	FRC (Abhängig vom internen A/D RC Oszillator)																											
5-3	CHS2:CHS0	Analoge Eingangs Auswahl-Bits <table><tr><th>BIT</th><th>Eingang</th></tr><tr><td>000</td><td>0 – AN0</td></tr><tr><td>001</td><td>1 – AN1</td></tr><tr><td>010</td><td>2 – AN2</td></tr><tr><td>011</td><td>3 – AN3</td></tr><tr><td>100</td><td>4 – AN4</td></tr><tr><td>101</td><td>5 – AN5</td></tr><tr><td>110</td><td>6 – AN6</td></tr><tr><td>111</td><td>7 – AN7</td></tr></table>	BIT	Eingang	000	0 – AN0	001	1 – AN1	010	2 – AN2	011	3 – AN3	100	4 – AN4	101	5 – AN5	110	6 – AN6	111	7 – AN7									
BIT	Eingang																												
000	0 – AN0																												
001	1 – AN1																												
010	2 – AN2																												
011	3 – AN3																												
100	4 – AN4																												
101	5 – AN5																												
110	6 – AN6																												
111	7 – AN7																												
2	GO/¬DONE	A/D Umwandlung Status Bit Um die Umwandlung zu starten wird das Bit auf 1 gesetzt. Wenn es auf 0 zurückgesetzt wird ist die Umwandlung fertig.																											
1	nicht genutzt	Wird als 0 gelesen.																											
0	ADON	A/D Power On Bit <table><tr><th>Bit</th><th>Auswirkung</th></tr><tr><td>1</td><td>Das A/D Konverter Modul wird aktiviert (angeschaltete).</td></tr></table>	Bit	Auswirkung	1	Das A/D Konverter Modul wird aktiviert (angeschaltete).																							
Bit	Auswirkung																												
1	Das A/D Konverter Modul wird aktiviert (angeschaltete).																												

		0	Das A/D Konverter Modul wird deaktiviert (ausgeschaltet) und versorgt keine anliegenden Umwandlungen mehr.
--	--	---	--

ADCON1 Register

Bit 7	6	5	4	3	2	1	Bit 0
ADFM	ADCS2	-	-	PCFG3	PCFG3	PCFG3	PCFG3

Konfiguration der einzelnen Bits:

Stelle	Bezeichnung	Bedeutung																																											
7	ADFM	A/D Ergebnis Formatierung Auswahl Bit																																											
		<table><tr><th>Bit</th><th>Auswirkung</th></tr><tr><td>1</td><td>„Rechts Bündig“ Die 6 Most Significant Bits des ADRESH Registers sind 0.</td></tr><tr><td>0</td><td>„Links Bündig“ Die 6 Least Significant Bits des ADRESL Regsitors sind 0.</td></tr></table>	Bit	Auswirkung	1	„Rechts Bündig“ Die 6 Most Significant Bits des ADRESH Registers sind 0.	0	„Links Bündig“ Die 6 Least Significant Bits des ADRESL Regsitors sind 0.																																					
		Bit	Auswirkung																																										
		1	„Rechts Bündig“ Die 6 Most Significant Bits des ADRESH Registers sind 0.																																										
0	„Links Bündig“ Die 6 Least Significant Bits des ADRESL Regsitors sind 0.																																												
Auf das genaue Auslesen wird weiter unten noch eingegangen.																																													
6	ADCS2	A/D Taktgeber für Umwandlungszeit-Auswahl Bits Es gilt die gleiche Tabelle und Aufteilung wie für Register ADCON0 oben beschrieben.																																											
5-4	nicht genutzt	Werden als 0 gelesen.																																											
3-0	PCFG3:PCFG0	A/D Port Konfigurations-Bits.																																											
		In der Tabelle werden nur die relevanten und im Moment benötigten Modi als Auszug aufgelistet. Die Tabelle kann auch in dem Manual des PIC18F4x2 nachgelesen werden.																																											
		Es kann auch Anstatt Vcc eine andere Referenzspannung angegeben werden. Wenn dies benötigt wird sollte im Manual des PIC18F4x2 nachgelesen werden. Alle hier aufgelisteten Varianten haben Vcc als Referenzspannung.																																											
		<table><tr><th>PCFG 3:0</th><th>AN7</th><th>AN6</th><th>AN5</th><th>AN4</th><th>AN3</th><th>AN2</th><th>AN1</th><th>AN0</th></tr><tr><td>0000</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr><tr><td>0010</td><td>D</td><td>D</td><td>D</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr><tr><td>0100</td><td>D</td><td>D</td><td>D</td><td>D</td><td>A</td><td>D</td><td>A</td><td>A</td></tr><tr><td>1110</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>A</td></tr></table>	PCFG 3:0	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	0000	A	A	A	A	A	A	A	A	0010	D	D	D	A	A	A	A	A	0100	D	D	D	D	A	D	A	A	1110	D	D	D	D	D	D
PCFG 3:0	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0																																					
0000	A	A	A	A	A	A	A	A																																					
0010	D	D	D	A	A	A	A	A																																					
0100	D	D	D	D	A	D	A	A																																					
1110	D	D	D	D	D	D	D	A																																					

		A = Analogger Eingang D = Digitaler I/O
--	--	--

Grundsätzlich gilt zu Beachten, dass bei einem Board RESET alle Pins die als Analoge Eingänge genutzt werden auch als Analoge Eingänge konfiguriert werden.

ADRESH und ADRESL Register

Es sollte erwähnt werden, dass die Werte die nach einer Umwandlung in diesen Register stehen bei einem Power-Reset nicht erhalten bleiben. Nach einem Power-Reset stehen undefinierte Werte in den Registern.

Wie man oben aus der Tabelle des ADCON1 Register erkennen kann, kann man die Formatierung des 10-Bit Ergebnisses in den Registern ADRESH und ADRESL beeinflussen.

Folgende Grafik macht dies deutlich:

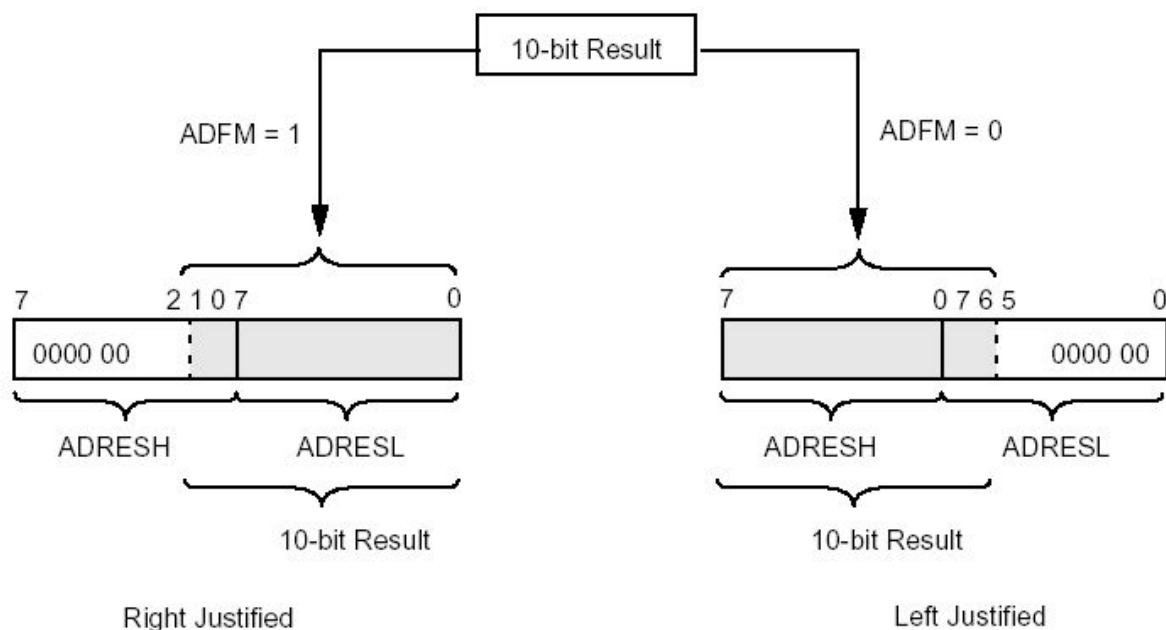


Bild 16 - A/D Wandler - Formatierung in den ADRESH und ADRESL Registern

Anhand der Grafik sollte erkennbar sein, dass man selbst entscheiden kann wie man das Ergebnis in den Registern formatiert haben möchte.

4.5.3 Ansteuerung in C

Nachdem nun die Register ausführlich erklärt sind sollte die Ansteuerung nun kein großes Problem mehr darstellen.

Allerdings gibt es ein paar Kleinigkeiten die man doch erwähnen sollte da diese, wenn man sie nicht beachtet, zu unerwünschten und nicht erklärbaren Auswirkungen führen.

Folgende Angaben werden beachtet:

- AN0 soll genutzt werden
- Fosc/32 Clock
- Das Ergebnis soll das LSB im LSB des ADRESL Registers haben

Zur Vereinfachung des Bearbeitens von dem 10-Bit Ergebnis wurde folgender Datentyp eingeführt:

```
// Datentypen festlegen
typedef unsigned char BYTE;      // 8-bit
typedef unsigned short int WORD; // 16-bit

// um ein 16-Bit Wort mit 2 8-Bit Wörtern schreiben zu können
typedef union _WORD_VAL
{
    WORD Val;
    BYTE v[2];
} WORD_VAL;
```

Falls manch einem nicht ganz klar ist weshalb dies eine Vereinfachung sein sollte hier ein kleiner Exkurs in C:

UNION

Mit dem Schlüsselwort UNION legt man fest, dass sich mehrere Variablen den gleichen Speicherplatz teilen.

In unserem konkreten Fall wurde erst eine 16-Bit Variable `WORD Val` und dann ein Array `BYTE v[2]` von jeweils 8-Bit angelegt. Das hat den Vorteil, dass man in 2 Schritten die jeweiligen 8-Bit Ergebnisse schreiben kann (in das Array) aber mit einem Zugriff das 16-Bit Wort lesen kann. Die genaue Anwendung dieses Vorteils wird gleich in dem C-Auszug sichtbar.

```
/* Zur Erstellung des 10-Bit Wortes */
WORD_VAL ADCResult;

/* AN0 Eingang, Fosc/32 clock */
ADCON0 = 0b10000001;

/* Analog Digitalwandler initialisieren */
ADCON1 = 0b10001110; // Das 10-Bit Wort soll von "Rechts" nach "Links" geschrieben werden
                      // AN0 wird zu einem Analogen Eingang geschaltet
```

ADCON0:

Für Fosc/32 müssen Bit 7 und 6 die Bitkombination 10 erhalten.

Um den A/D Wandler zu starten muss das Power-On Bit 0 auf 1 gesetzt werden.

ADCON1:

Damit das LSB des Results im LSB des ADRESL Register steht muss Bit 7 auf 1 gesetzt werden.

Da Fosc/32 genutzt werden soll wird Bit 6 auf 0 gesetzt.

Die Bits 3 bis 0 haben die Bitkombination 110 was bedeutet, dass nur der Pin 0 für den Eingang AN0 als analoger Eingang verwendet wird.

Da nun die Konfiguration erst einmal geschrieben werden muss bevor die Umwandlung gestartet werden kann sollte man direkt danach eine kleine Verzögerung einbauen. In unserem Fall wird dies folgendermaßen gelöst:

```
/* Abwarten bis Konfiguration geschrieben */
ADCResult.v[0] = 100;
```

```
while(ADCResult.v[0]--);
```

Nun kann die Umwandlung starten:

```
/* Konvertierung des AN0 Eingang */  
ADCON0bits.GO = 1;           // Konvertierung starten  
while(ADCON0bits.GO);        // Wenn Konvertierung fertig -> ADON0_GO = 0
```

Man kann hier gut das Abwarten bis die Umwandlung fertig ist erkennen.

Jetzt kommt der Teil bei dem man den Vorteil der UNION erkennen kann. In PDV wurde in einer Praktikumsaufgabe ein 16-Bit Timer verwendet bei dem man die zwei 8-Bit Teilzeiten zusammensetzen musste. Damals wurde noch fleißig geshiftet und gebastelt, damit am Ende dann ein 16-Bit Wort herauskam. Die jetzt gezeigte Lösung finde ich persönlich wesentlich eleganter.

```
/* 8-Bit Werte auf ein 16-Bit Bereich schreiben */  
ADCResult.v[0] = ADRESL;  
ADCResult.v[1] = ADRESH;  
  
/* 10-Bit Wert in ASCII String konvertieren */  
itoa(ADCResult.Val, AN0String);
```

Man kann erkennen, dass die beiden Teilergebnisse aus den ADRESL und ADRESH Registern als 8-Bit Worte in das Array geschrieben werden.

Zur Umwandlung für die Ausgabe allerdings kann man dann ganz einfach auf das 16-Bit Wort zugreifen und muss nicht erst lange herumbasteln.

In diesem Beispiel wird das Ergebnis in einen String gespeichert um später auf dem LCD-Display ausgegeben zu werden. Dies kann man im nächsten Kapitel in den Tutorials ausführlich nachlesen.

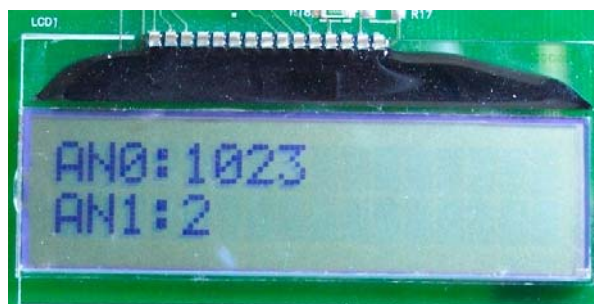


Bild 17 - AN0 und AN1 Potentiometer mit Ausgabe auf dem LCD-Display

Es sollte an dieser Stelle noch kurz erwähnt werden, dass die Beschriftung der Potentiometer verkehrt herum ist. Die Bezeichnung AN0 gilt dem unteren Potentiometer und die Bezeichnung AN1 dem oberen. Diese Erkenntnis zu erlangen hat uns mehrere Stunden gedauert, da wir der Einfachheit halber erst AN0 ansprechen wollten dieser aber, wenn man an den AN0 Potentiometer dreht, keinerlei Veränderung zeigt. Nach längerem suchen und debuggen stellte sich heraus, dass einfach nur die Beschriftung verkehrt herum aufgedruckt wurde.

Nun ist die Umwandlung des analogen Wertes an AN0 fertig und auch recht einfach zu bewältigen. Interessanter wird es aber bei dem AN1 Eingang und da auf dem

PICDEM.net Demo Board auch ein Potentiometer für AN1 verfügbar ist wird dieser hier an dieser Stelle auch noch besprochen.

Da es keinen Modus gibt bei dem nur AN1 oder zusammen mit AN0 als analogen Eingang zu verwenden und der Rest digitale I/O Eingänge sind, muss der nächst Beste Modi genommen werden. Bei dem nächst Besten Modus wird der Eingang AN3 auch noch auf einen analogen Eingang geschaltet.

Da bei dem PICDEM.net Demo Board Port A für die Analogen Eingänge sowie für die LEDs und das LCD-Display verwendet wird ist es notwendig dass bis auf AN0 und AN1 alle anderen Pins digitale I/O Eingänge sind. Es kann sonst zu merkwürdigen Ergebnissen bei den LEDs und dem LCD-Display kommen.

Es ist daher zwingend notwendig nach dem Umwandeln des analogen Wertes an dem Eingang AN1 sofort wieder die Pins auf digitale I/O Eingänge zurück zu schalten.

Für die Umwandlung an AN1 gelten die gleichen Angaben wie bei der eben besprochenen Umwandlung an AN0.

```
/* Right justified - von "rechts" nach "links" lesend, RA3 -> analog */
ADCON1 = 0b10000100;

/* AN1 Eingang wählen -> Bit 5-3, Fosc/32 -> Bit 7-6 */
ADCON0 = 0b10001001;

/* Abwarten bis Konfiguration geschrieben */
ADCResult.v[0] = 100;
while(ADCResult.v[0]--);

/* Konvertierung des AN1 Eingang */
ADCON0bits.GO = 1; // Konvertierung starten
while(ADCON0bits.GO); // Wenn Konvertierung fertig -> ADON0_GO = 0

/* 8-Bit Werte auf ein 16-Bit Bereich schreiben */
ADCResult.v[0] = ADRESL;
ADCResult.v[1] = ADRESH;

/* 10-Bit Wert in ASCII String konvertieren */
itoa(ADCResult.Val, AN1String);

/* Zurücksetzen des RA3 Pins */
ADCON1 = 0b10001110; // RA0 -> analoger Input
```

Ich denke der Code mit Kommentaren ist selbsterklärend und braucht an dieser Stelle nicht weiter kommentiert zu werden.

Alle hier aufgelisteten Registertabellen kann im PIC18F4x2 Manual nachgelesen werden. Dieses ist auf der CD im Ordner /pdf/ verfügbar.

4.6 Interrupts

4.6.1 Eigenschaften

Der PIC18F452 verfügt über die Möglichkeit Interrupts zweier Prioritätsstufen zu bearbeiten. Interrupts niedriger Priorität können nur von Interrupts hoher Priorität unterbrochen werden. Hochpriorität Interrupts können nicht unterbrochen werden. Tritt ein Interrupt auf so werden automatisch alle Interrupts gesperrt. Der Controller kann Interrupts von verschiedenen Quellen bearbeiten:

- Timer0 - 3
- PortB
- USART (Rc und Tx)

- 3 Externe Leitungen
- A/D Wandler
- CPP1 und 2
- EEPROM/Flash Schreiboperationen
- Low Voltage Detection

Alle Interrupt spezifischen Einstellungen, wie Wahl der Priorität und Maskieren der Interrupts, sowie alle Ermittlungen welches Gerät einen Interrupt ausgelöst hat, werden durch den Zugriff auf zehn Register realisiert.

- Interrupt Control Register 1(INTCON)
- Interrupt Control Register 2(INTCON2)
- Interrupt Control Register 3(INTCON3)
- Peripheral Interrupt Request Register 1(PIR1)
- Peripheral Interrupt Request Register 2(PIR2)
- Peripheral Interrupt Enable Register 1(PIE1)
- Peripheral Interrupt Enable Register 2(PIE2)
- Peripheral Interrupt Priority Register 1(IPR1)
- Peripheral Interrupt Priority Register 2(IPR2)
- Reset Control Register(RCON)

Auf die einzelnen Registertabellen soll an dieser Stelle nicht eingegangen werden, da es sonst den Rahmen sprengen würde. Allerdings können diese auch im PIC18Fxx2 Manual, das auf der CD im Verzeichnis /pdf/ zu finden ist, eingesehen werden.

4.6.2 Implementieren einer Interrupt Service Routine in C

Wird ein Interrupt ausgelöst muss die Quelle des Interrupts von der Software ermittelt werden, denn die Hardware unterscheidet nur zwischen High und Low Priority Interrupts. Entsprechend existieren auch nur zwei Adressen im Programmspeicher, von wo aus eine Funktion aufgerufen wird. An der Adresse 0x0008 sitzt der High Priority Vector und an der Adresse 0x0018 sitzt der Low Priority Vector. An diesen Stellen muss nun eine Funktion gespeichert werden, die dann die jeweilige ISR aufruft. Dies bewerkstelligt man mit dem Pragma code.

```
#pragma code high vector = 0x08
void interrupt at high vector(void)
{
    if(INTCONbits.INT0IF == 1) // Externer Int 0
    {
        asm GOTO DecoderISR endasm // Aufrufen der ISR per Assemblerbefehl GOTO
    }
    else if(INTCONbits.TMR0IF == 1) // Timer 0 Int
    {
        _asm GOTO Timer0ISR _endasm
    }
    else if(PIR1bits.RCIF == 1) // Recv Int
    {
        asm GOTO USARTrxISR endasm
    }
}
#pragma code // Ab hier kann wieder der Linker bestimmen wie gespeichert wird
```

In dieser Funktion ermittelt man per Zugriff auf die jeweiligen Register, welches Gerät nun einen Interrupt ausgelöst hat und ruft die ISR dann mit dem Assembler Befehl GOTO auf. Eine Interrupt Service Routine wird mit dem Pragma interrupt als solche gekennzeichnet. Dies hat zur Folge, dass beim Aufruf der Routine der

Programmkontext auf einem Fastreturn Stack gespeichert wird und alle Interrupts gesperrt werden.

```
#pragma interrupt DecoderISR
void DecoderISR(void)
{
    direction = PORTBbits.RB1;
    tick counter++;
    INTCONbits.INT0IF = 0; // Interrupt Flag löschen
}
```

Eventuell müssen die Interrupt Flags softwareseitig in den jeweiligen Registern gelöscht werden.

5 Beispielprogramme – Tutorials

5.1 Grundsätzliches

Das hier ein Kapitel mit Tutorials entsteht entspricht einfach nur der Tatsache, dass wir im Laufe unsres Projekts keinerlei richtige Einführung in die Software MPLAB v.6.20 gefunden haben. Es gibt kleinere Projekte oder auch Minimale Dokumentationen die mitgeliefert wurden aber entweder sie gilt der gänzlich anders Aufgebauten MPLAB Version 5.x oder der Umgang mit der Software beschränkte sich auf „Verwendet wurde die Software MPLAB v6.x“.

Da die Version 6.x erst gerade auf den Markt kam als wir angefangen haben gab es auch daher schon sehr wenig Dokumentation darüber. Über die Vorgänger Versionen 5.x gibt es sehr viel aber leider unterscheiden sich die beiden Versionen im Menü und Aufbau sehr, so dass man diese Beschreibungen nicht verwenden konnte.

Zusätzlich gibt es eine kleine Einführung in den Projektaufbau, die Notwendigen Einstellungen sowie das Debuggen.

Diese gesamten Einführungen werden einmal Ausführlich in dem Kapitel „A/D-Wandler und Anzeige auf dem LCD-Display“ erläutert, da das eigentliche Programmieren bei diesem Projekt sehr gering ausfällt.

5.2 AD-Wandler und Anzeige auf dem LCD-Display

5.2.1 Beschreibung

Dieses Projekt diene uns selber am Anfang, nach langwierigen Untersuchen des PICDEM.net Demo Board und der Peripherie, endlich ein paar Funktionen des Microcontrollers sowie das LCD-Display zusammen zu benutzen.

Die Aufgabe, die wir uns dabei gestellt haben war die zwei verfügbaren Potentiometer des PICDEM.net Demo Board, die an die beiden Analog/Digital-Wandler Eingänge AN0 und AN1 angeschlossen sind, zu nutzen, deren Werte umzuwandeln und anschließend auf dem LCD-Display lokal auszugeben.

Als kleines Gimmick werden die LEDs genutzt. Sobald sich ein analog anliegender Wert ändert und der A/D Wandler wieder konvertiert soll die entsprechende LED kurz aufblinken.

Es soll nun folgende Peripherie genutzt werden:

- 2 Potentiometer auf dem PICDEM.net Demo Board
- Analog/Digital-Wandler Modul
- LCD-Display
- User defined LEDs

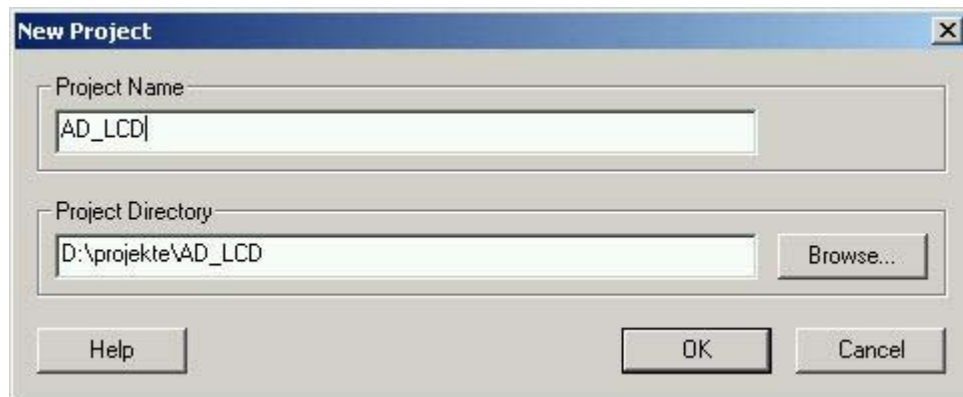
Das Projekt soll in C geschrieben werden und die verfügbare Entwicklungsumgebung von MPLAB kennen gelernt und genutzt werden.

Zudem soll einer der großen Vorteile, das In-Circuit-Debuggen, durch das ICD 2 genutzt werden.

5.2.2 Projekt erstellen und konfigurieren

Für das Projekt muss eine Projektumgebung erstellt werden. Dies geschieht entweder mit dem Project Wizard oder Manuell. Wir werden dies hier manuell durchgehen, da man dann weiß wo man nachschauen muss wen man zu einem späteren Zeitpunkt etwas ändern möchte.

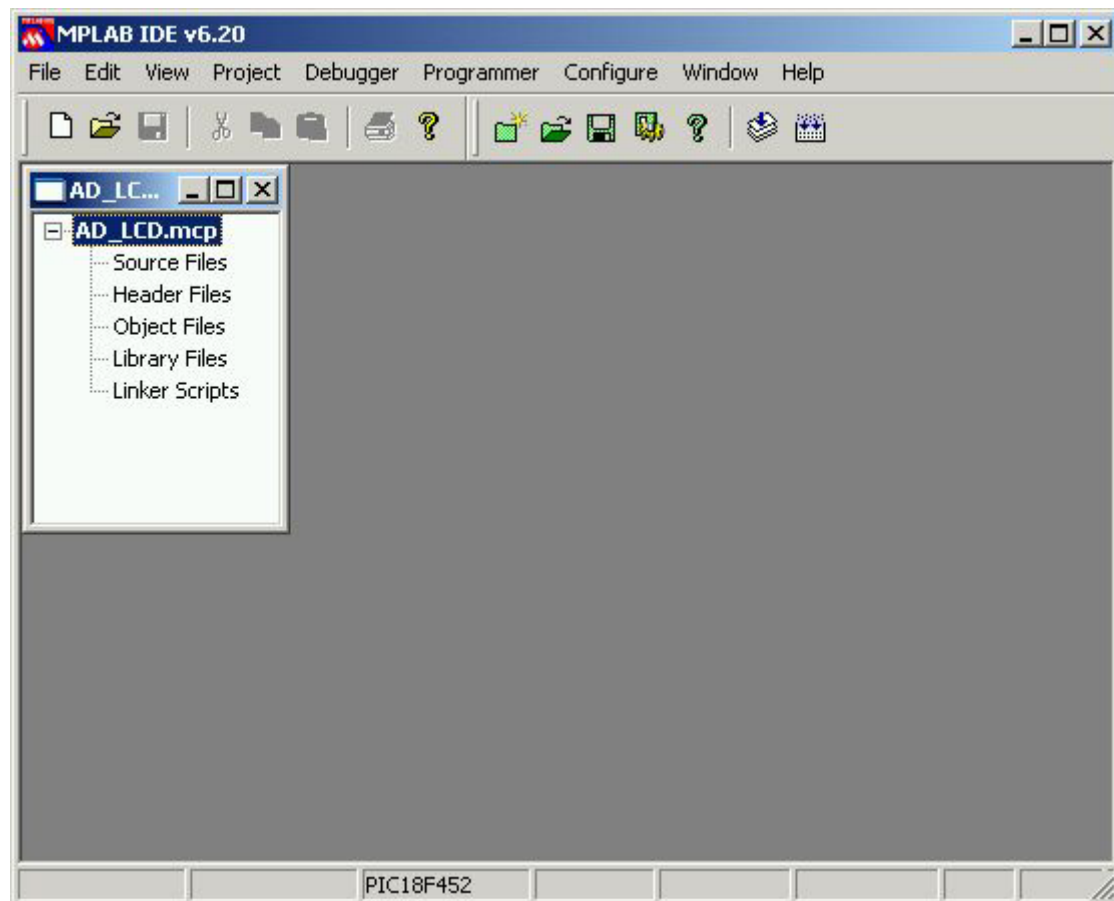
Neues Projekt erstellen: Menü: Project->New...



Hierbei gilt zu Beachten, dass der Pfad, in dem das Projekt liegt, die Zeichenlänge von 65 Zeichen nicht überschreitet, da sonst später der Compiler wegen zu großer Zeichenlänge abbricht.

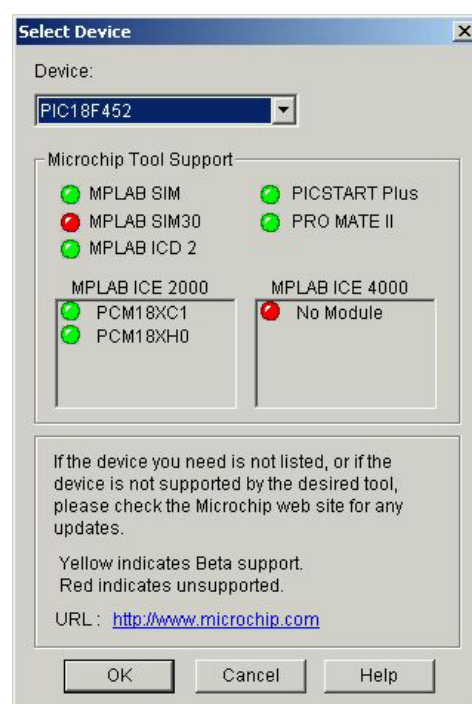
Zudem müssen die verwendeten Ordner manuell im Datei Explorer erstellt werden, da dieser Wizard das nicht übernimmt. Er kann nur in schon bestehende Verzeichnisse Projekte erstellen.

Es sollte nun folgende Projektumgebung zu sehen sein:



Auf der linken Seite kann man die Projektübersicht sehen. In diesem Fenster sind alle verwendeten Dateien aufgelistet, so dass man schnellen Zugriff und eine Übersicht hat.

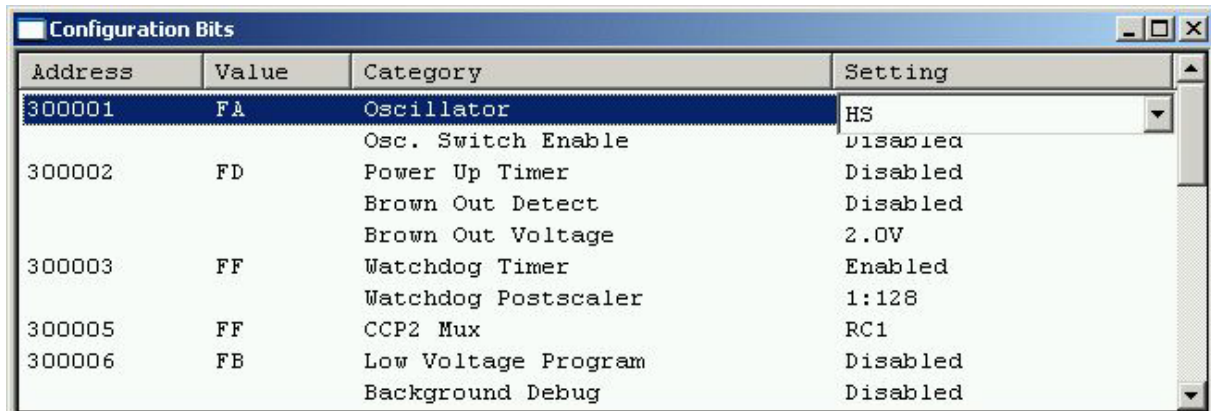
Nachdem das Projekt erstellt wurde muss der verwendete Microcontroller angegeben werden: Menü: Configure->Select Device...



Wir verwenden natürlich den PIC18F452 Microcontroller. Auf diesem Dialog kann man zudem noch die Informationen erhalten welche Tools dieser Microcontroller unterstützt.

Für uns wichtig an dieser Stelle ist die grüne Diode bei MPLAB ICD 2 den wir als Programmer und Debugger verwenden wollen.

Nun sollten noch der verwendete Oszillator angegeben werden: Menü:
Configure->Configuration Bits...



Address	Value	Category	Setting
300001	FA	Oscillator	HS
		Osc. Switch Enable	Disabled
300002	FD	Power Up Timer	Disabled
		Brown Out Detect	Disabled
		Brown Out Voltage	2.0V
300003	FF	Watchdog Timer	Enabled
		Watchdog Postscaler	1:128
300005	FF	CCP2 Mux	RC1
300006	FB	Low Voltage Program	Disabled
		Background Debug	Disabled

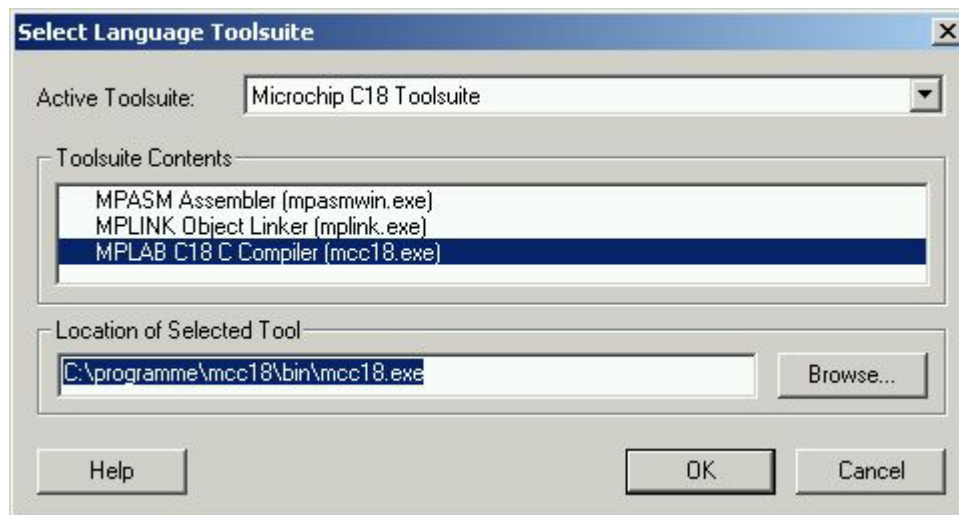
Sowie den „Brown Out Detect“ und „Low Voltage Program“ auf „Disabled“ setzen da diese default mäßig auf „Enabled“ stehen aber bei uns nicht gebraucht werden.

Das einzige was später noch einmal geändert wird ist „Background Debug“ auf „Enabled“ setzen um debuggen zu können.

Als nächstes muss die Sprache gewählt werden mit der man programmieren möchte. Da wir die Hochsprache C verwenden wollen wird an dieser Stelle vorausgesetzt, dass dies auch ordnungsgemäß installiert ist. Die Installation des C18 Compilers ist recht einfach und am Ende wird man gefragt ob er selber die Pfade für MPLAB setzen möchte, was man der Einfachheit halber mit „Ja“ antworten sollte.

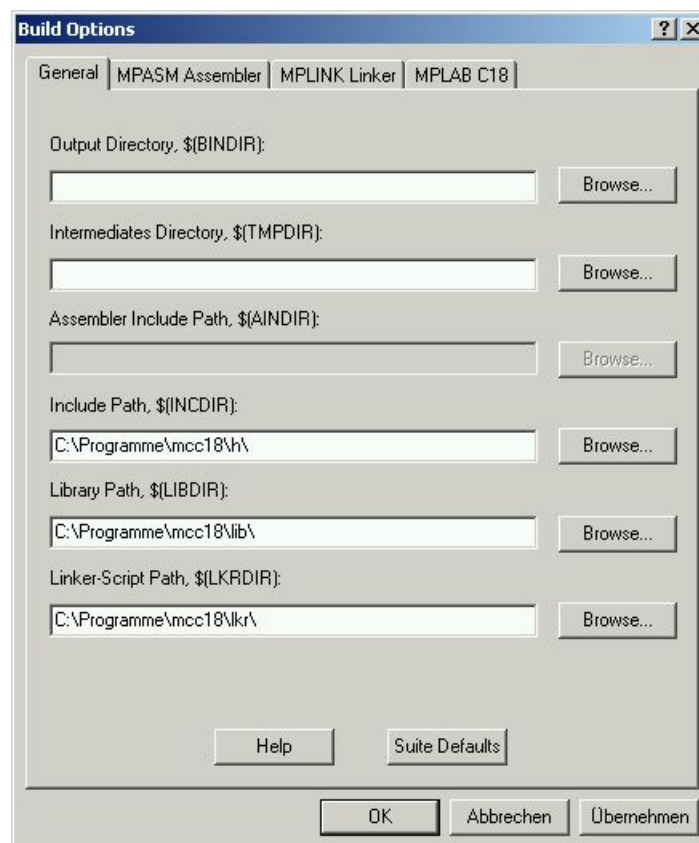
Falls dies aber mit „Nein“ geantwortet wurde kann man auch noch später die Pfade zu dem C18 Compiler setzen: Menü: Project->Set Language Tool Locations...

Nun muss aber im Projekt selber noch angegeben werden mit welcher Sprache programmiert werden soll: Menü: Project-> Select Language Toolsuite...



Nachdem nun die Sprache gewählt wurde müssen der Include Pfad, der Library Pfad sowie der Linker Pfad gesetzt werden:

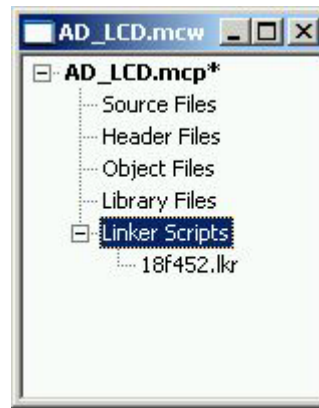
Menü: Project->Build Options...->Project



Zusätzlich ist eine Pfadangabe für den Output möglich. Wenn dieser nicht gesetzt wird ist der Output default mäßig der Projekt Ordner.

Jetzt fehlt nur noch eine Einstellung und dann sind auch schon alle Grundsätzlichen Projekteinstellungen getätigt.

Es muss per Hand das gewünschte Linker-Script angegeben werden. In dem Übersichtsfenster ADLCD.mcw werden alle genutzten Files aufgelistet. In diesem Fenster klickt man mit der rechten Maustaste auf „Linker Scripts“ -> Add Files...



Es gibt für den PIC18F452 zwei verschiedene Linker-Scripte. Der eine ist der hier in der Grafik abgebildete und der andere hat am Ende von seinem Namen ein i (18f452i.lkr). Laut einem Tutorial soll dieser speziell für Debug Zwecke gut sein.

Ein großer Nachteil der MPLAB IDE ist, dass sie sich nicht global seine Einstellungen wie Header-Files-, Bibliotheks- oder Linker-Verzeichniss merkt. Sobald man nun also ein neues Projekt erstellt muss man die gesamte Prozedur der Einstellungen wiederholen. Ein Nachteil hat dies auch bei Projekten die dann auf einem anderen Rechner weiterentwickelt werden soll denn die ganzen Pfade werden sich für das Projekt auf der Maschine gemerkt wo es erstellt wurde. Möchte man also das Projekt auf einem anderen Rechner weiterentwickeln müssen erst alle Einstellungen geändert werden was sehr mühsam ist.

Eine Verbesserung wäre also sich das auf Programmebene und nicht auf Projektebene zu merken. Aber da es ja viele Möglichkeiten, z.B. in der Programmiersprache, gibt wird das wohl auch irgendwo einen Sinn haben, dass alles im Projekt lokal gespeichert wird.

Nun sind die Rahmenbedingungen für das Projekt erstellt und eingerichtet, so dass man nun zum eigentlichen Projekt und dem programmieren kommt.

5.2.3 Source Code

Wenn man nun durch Menü: File->New... sich eine neue Datei erstellt und diese dann abspeichert ist sie leider noch nicht im Projekt integriert. Man muss manuell seine Files dem Projekt hinzufügen (Rechte Maustaste im Projektübersichtsfenster). Erst dann gehören sie offiziell dem Projekt an und werden beim kompilieren und verlinken beachtet.

Der eigentliche Source-Code kann auf der CD im Ordner Projekte gefunden werden. Hier wird jetzt nur die Datei main.c aufgelistet, da sie das ganze Herzstück hält.

Zudem wird hier nicht auf jede Zeile des Source-Codes eingehen, da wir das Grundwissen von C und der Handhabung dieser als gegeben voraussetzen. Somit wird hier lediglich der Quellcode abgedruckt und das grundsätzliche Prinzip der Vorgehensweise erklärt. Vielmehr ist später das debuggen interessant was auch intensiver behandelt wird.

Das Prinzip wird nach dem Listing des Source-Codes beschrieben.

Source-Code: main.c

```
#include "defines.h"
#include <delays.h>
#include <string.h>
#include "xlcd.h"
#include "delay.h"

/* Für Ausgabe der gewandelten Werte */
char AN0String[8] = ""; // Für Ausgabe AD0
char AN1String[8] = ""; // Für Ausgabe AD1

/* AN0 und AN1 wandeln */
void convertAD(void);

void main(void)
{
    /* Zur Überprüfung ob sich etwas geändert hat - nur dann Ausgabe */
    char oldAN0[8] = ""; // Für Ausgabe AD0
    char oldAN1[8] = ""; // Für Ausgabe AD1

    /* Ausgabe Text */
    char an0String[5] = "AN0:";
    char an1String[5] = "AN1:";

    /* Initialisierung */

    /* LCD Display initialisieren */
    XLCDInit();
    XLCDClear();

    while(1) {
        convertAD();

        // LCD Display löschen
        XLCDClear();

        XLCDGoto(0,0);
        // Anfangstext für AD0
        while(XLCDIsBusy());
        XLCDPutString(an0String);

        // AN0 Analogen Wert rausschreiben
        while(XLCDIsBusy());
        XLCDPutString(AN0String);

        XLCDGoto(1,0);
        // Anfangstext für AD0
        while(XLCDIsBusy());
        XLCDPutString(an1String);

        // Analogen Wert rausschreiben
        while(XLCDIsBusy());
        XLCDPutString(AN1String);

        /*
         * AN0-Eingang
         * Prüfen ob sich seit der letzten Wandlung etwas verändert hat.
         * Wenn ja, dann Daten neu auf LCD schreiben und Diode aufleuchten lassen
         */
        if(strcmp(AN0String, oldAN0) != 0){
            // neuen Wert speichern
            strcpy(oldAN0, AN0String);

            // Diode aufleuchten lassen
            TRISAbits.TRISA2 = 0; // Output
            PORTAbits.RA2 = 0; // An
            DelayXMs(20);
            PORTAbits.RA2 = 1; // Aus
            TRISAbits.TRISA2 = 1; // Input
        }

        /*
         * AN1-Eingang
         * Prüfen ob sich seit der letzten Wandlung etwas verändert hat.
         * Wenn ja, dann Daten neu auf LCD schreiben und Diode aufleuchten lassen
         */
    }
}
```

```

        */
        if(strcmp(AN1String, oldAN1) != 0){
            // neuen Wert speichern
            strcpy(oldAN1, AN1String);

            // Diode aufleuchten lassen
            TRISAbits.TRISA3 = 0; // Output
            PORTAbits.RA3 = 0; // An
            DelayXMs(20);
            PORTAbits.RA3 = 1; // Aus
            TRISAbits.TRISA3 = 1; // Inputt
        }

        // Damit LCD nicht flackert - 500ms warten
        DelayXMs(500);
    }

    return;
}

void convertAD(void) {

    /* Zur Erstellung des 10-Bit Wortes */
    WORD VAL ADCResult;

    /* AN0 Eingang, Fosc/32 clock */
    ADCON0 = 0b10000001;

    /* Analog Digitalwandler initialisieren */
    ADCON1 = 0b10000000; // Das 10-Bit Wort soll von "Rechts" nach "Links" geschrieben
    werden

    /* Abwarten bis Konfiguration geschrieben */
    ADCResult.v[0] = 100;
    while(ADCResult.v[0]--);

    /* Konvertierung des AN0 Eingang */
    ADCON0bits.GO = 1; // Konvertierung starten
    while(ADCON0bits.GO); // Wenn Konvertierung fertig -> ADON0 GO = 0

    /* 8-Bit Werte auf ein 16-Bit Bereich schreiben */
    ADCResult.v[0] = ADRESL;
    ADCResult.v[1] = ADRESH;

    /* 10-Bit Wert in ASCII String konvertieren */
    itoa(ADCResult.Val, AN0String);

    /*
    * Konvertierung des AN1-Eingang
    *
    * Auszug aus MHCP-Stack Source:
    * Am PICDEM.net Board sollten RA2-RA7 digital sein sonst funktionieren
    * LED, LCD und NIC nicht korrekt.
    * Da es keinen Mode gibt wie dem AN0 und AN1 analoge Inputs und der Rest
    * digitale Pins sind, wir vorübergehend RA3 während der AD-Wandlung
    * analog geschaltet. Wenn die AD-Wandlung um ist wird RA2 wieder zurück
    * auf digital geschaltet.
    */
    /* Right justified - von "rechts" nach "links" lesend, RA3 -> analog */
    ADCON1 = 0b10000100;

    /* AN1 Eingang wählen -> Bit 5-3, Fosc/32 -> Bit 7-6 */
    ADCON0 = 0b10001001;

    /* Abwarten bis Konfiguration geschrieben */
    ADCResult.v[0] = 100;
    while(ADCResult.v[0]--);

    /* Konvertierung des AN1 Eingang */
    ADCON0bits.GO = 1; // Konvertierung starten
    while(ADCON0bits.GO); // Wenn Konvertierung fertig -> ADON0_GO = 0

    /* 8-Bit Werte auf ein 16-Bit Bereich schreiben */
    ADCResult.v[0] = ADRESL;

```

```
ADCResult.v[1] = ADRESH;

/* 10-Bit Wert in ASCII String konvertieren */
itoa(ADCResult.Val, AN1String);

/* Zurücksetzen des RA3 Pins */
ADCON1 = 0b10001110; // RA0 -> analoger Input
}
```

Da der Source-Code ausführlich kommentiert ist können die Erklärungen zu programmiertechnischen Fragen direkt aus dem Listing genommen werden. Die Ansteuerung des A/D-Wandlers und des LCD-Displays wird zudem ausführlich im Kapitel 4 „Ansteuerung der Peripherie“ beschrieben.

Das Grundsätzliche Prinzip des main-loops ist folgender Ablauf:

- convertAD()
 - umwandeln des Analogen Wertes an AN0
 - schreiben des umgewandelten Wertes in einen String
 - umwandeln des Analogen Wertes an AN1
 - schreiben des umgewandelten Wertes in einen String
- Ausgabe des Ergebnis-Strings von AN0 auf dem LCD-Display
- Ausgabe des Ergebnis-Strings von AN1 auf dem LCD-Display

Zudem werden noch Standardbibliotheken wie `<delay.h>` und `<string.h>` genutzt. Aber auch die eigene „`xlcd.h`“, die von dem MCHPStack übernommen und modifiziert wurde wird hier genutzt.

Weiter soll aber an dieser Stelle auf diesen Code nicht eingegangen sein, da es von programmiertechnischer Seite nicht allzu aufwendig ist und die Kommentare im Source-Code alles sehr genau erklären.

5.2.4 Programmieren des Microcontroller mit ICD 2

Jetzt kommt eine der Besonderheiten die das Entwickeln für den Microcontroller sehr vereinfachen. Bisher konnte man IC's nur programmieren indem man sie von ihrem Sockel geholt und auf ein Programmiergerät gesteckt hat, welches an der Parallelen Schnittstelle hing. Diese Methodik ist natürlich für das IC selber sehr aufwendig und abnutzend. Zudem ist die gesamte Prozedur des Programmierens sehr aufwendig und zeitraubend.

In Kapitel 2 wurde die Hardware beschrieben. Unter anderem auch das ICD 2 und seine Vorteile. Hier kommen diese Vorteile jetzt zum tragen. Das ICD 2 lässt sich über USB oder der seriellen Schnittstelle von Rechner aus ansteuern und ist mit dem PICDEM.net Board verbunden.

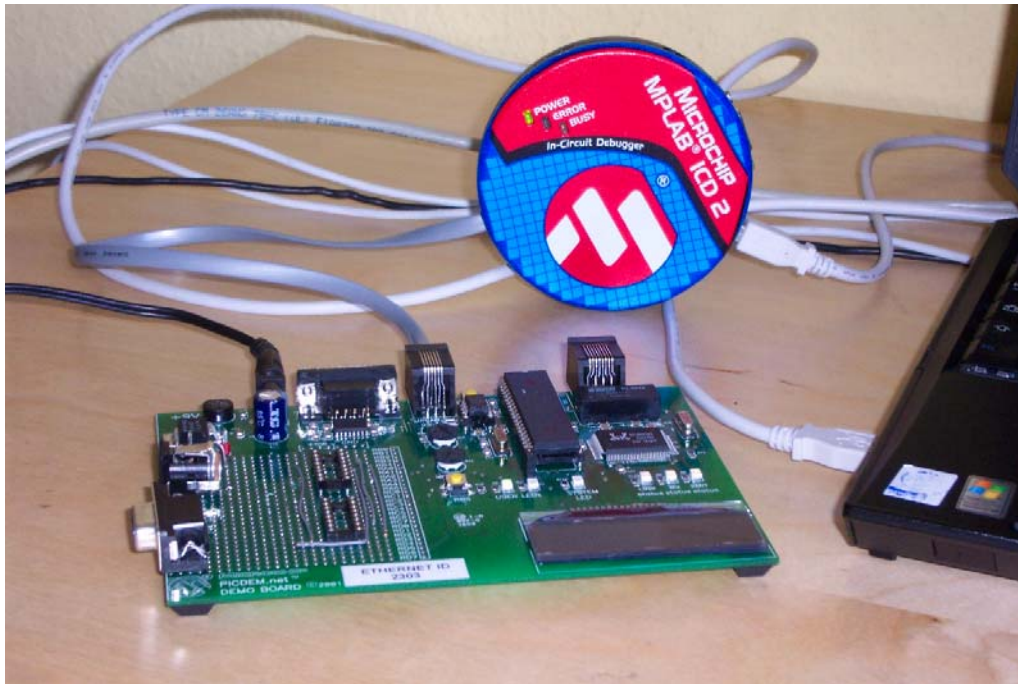


Bild 18 - ICD2 verbunden mit dem PICDEM.net Demo Board und Laptop

Auf dem Bild kann man recht gut erkennen wie die gesamte Hardware miteinander verbunden aussieht.

Wenn das ICD 2 ordnungsgemäß installiert ist sollte nach dem Auswählen des Programmiergeräts in der MPLAB IDE eine Connect-Meldung in der Output Box erscheinen.

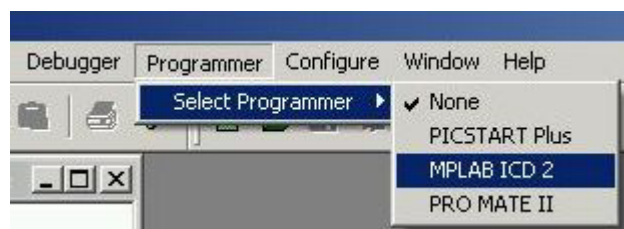


Bild 19 - Auswählen des Programmiergeräts

Falls sich an dieser Stelle Schwierigkeiten ergeben kann dies mehrere Gründe haben, die wir über die Zeit herausbekommen haben:

- ICD 2 USB Treiber wurde nicht richtig installiert. Eine Genaue Anleitung gibt es direkt nach der Installation der MPLAB IDE.
- Es wird nicht die neueste Version der MPLAB IDE verwendet. Im Moment Version 6.20. Vorherige Versionen haben Probleme mit USB 2.0.
- Die MPLAB IDE ist schon gestartet und das USB Kabel des ICD 2 wurde erst nachträglich eingesteckt. Das Gerät wird so oft nicht erkannt. Man muss hierbei die IDE noch einmal schließen und die ICD 2 USB Stecker erneut stecken. Erst dann wieder die IDE starten.

Wenn nun die ICD 2 korrekt erkannt wurde, kann der Microcontroller programmiert werden: Menü: Programmmer->Program

Folgende Meldung wird bei einer korrekten Programmierung im Output Fenster ausgegeben:

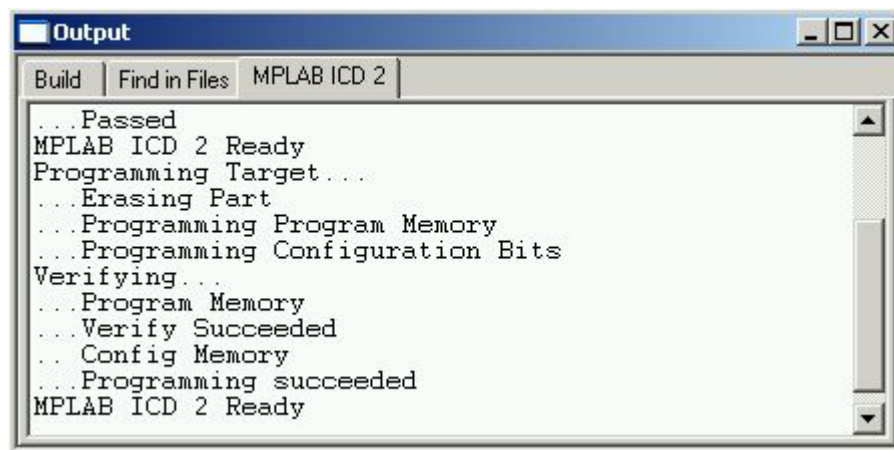
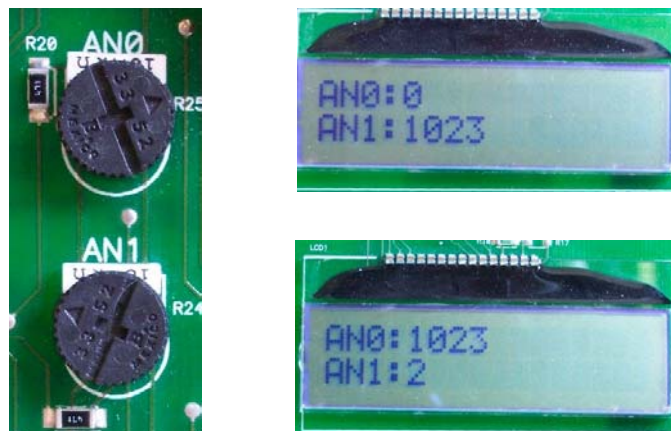


Bild 20 - Programmieren des Microcontrollers durch ICD 2

Wenn das Programmieren abgeschlossen ist kann das Kabel von ICD 2 zu dem PICDE.net Demo Board abgezogen werden und die Anwendung startet. Manchmal gibt die Anwendung an dieser Stelle nur nicht definierbares aus. In diesem Falle ist ein „Hardware-Reset“ notwendig, was soviel bedeutet wie „die Stromversorgung kurz unterbrechen“. In dem Fall, dass alles korrekt programmiert wurde sollte die Anwendung nun einwandfrei laufen wie es folgende Bilder zeigen.



5.2.5 In-Circuit-Debuggen mit dem ICD 2

Nun kommen wir zum absoluten Highlight des ICD 2. Es ist möglich zur Laufzeit Register auszulesen oder das Programm an bestimmte, durch Breakpoints definierte Stellen anzuhalten und Schritt- oder Zeilenweise vorwärts zu gehen. Diese Methodik ist sehr effektiv und ohne diese Debugmöglichkeit wären einige Untersuchungen noch viel aufwendiger gewesen.

Kommen wir nun zu den eigentlichen Einstellungen bei der MPLAB IDE:

- Configuration Bits – Enable Background Debug
- Debugger wählen

Am Anfang dieses Tutorials haben wir einige Configuration Bits Disabled – unter anderem den Background Debug. Dieses Configuration Bit muss nun wieder Enabled werden um das Debuggen zu ermöglichen. Diese Information ist für den Compiler deswegen notwendig, da er eigene Wörter und Sprünge eincompilieren muss.

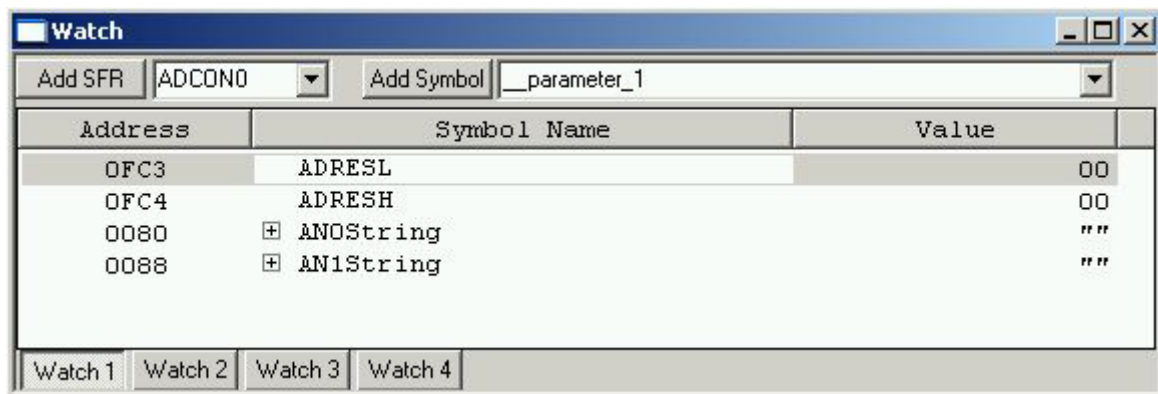
Zudem muss jetzt nicht nur der Programmer sondern auch der Debugger im Menü selektiert werden.

Menü: Debugger->Select Tool->MPLAB ICD 2

Da nun der Debugger ausgewählt und das Output Window einen korrekten Verbindungstest anzeigt kann nun das Debuggen ausprobiert werden.

Zuerst wird das Watch Window geöffnet in dem die Register ADRESL und ADRESH sowie die beiden Ergebnis Strings AN0String und AN1String angezeigt werden sollen.

Menü: View->Watch



Um in der Watch Register oder Variablen beobachten zu können müssen diese erst eingefügt werden. Da es sich bei den Registern ADRESL und ADRESH um Special Function Register handelt findet man diese in der linken Listbox direkt neben dem Button „Add SFR“. Dort sucht man diese, selektiert sie und drückt anschließend den Button „Add SFR“ um sie auf unsern Watch einzubinden.

Genauso verhält es sich mit unseren String Variablen wobei dies in der rechten Listbox zu finden sind.

Grundsätzlich gilt bei der Watch immer die benötigte Variable oder Register suchen, selektieren und „adden“. Dann sind sie auf der Watch sichtbar und können beobachtet werden.

Ein solches Watch-Window kann man abspeichern um nicht jedes mal, sofern man seine Projektumgebung nicht speichert, wieder von neuem suchen und einfügen muss.

Anhand der Quick-Button Leiste kann man, nachdem das Programm angehalten wurde Schrittweise vorwärts gehen. Hier aber erst eine kleine Erklärung des Menüs:

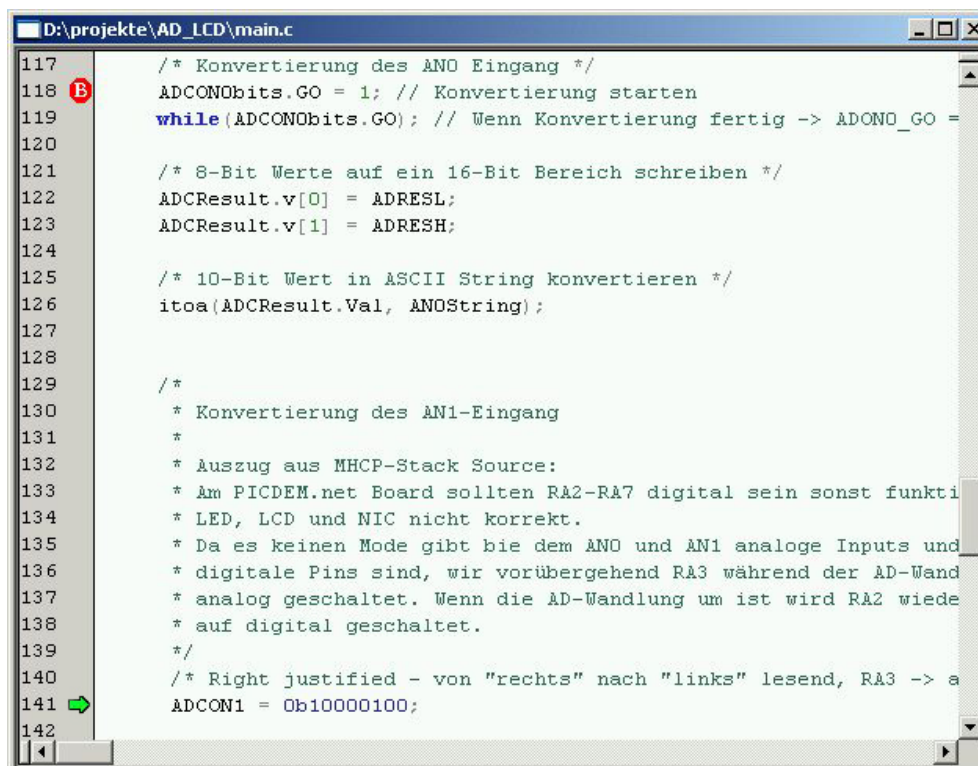


Von Links nach Rechts gibt es folgende Buttons:

- Run
- Stop
- Step – Führt den nächsten Befehl/Zeile aus. Bei einer Funktion wird in die Funktion gesprungen
- Step Over – Durchläuft die nächste Zeile und hält erst nach dieser an

Wir haben nun unseren Breakpoint in der Funktion `convertAD()` direkt vor der Umwandlung des AN0 Eingangs gesetzt. Nachdem wir das Programm nun im Debug Modus auf dem Microcontroller programmiert haben kann man es mit der Run Taste starten. Wenn man keinen Breakpoint gesetzt hat, kann man auch keinen Unterschied zu der normalen Microcontroller Programmierung erkennen. Die Anwendung würde dann durchgehend laufen – ohne Unterbrechen.

Da wir aber einen Breakpoint gesetzt haben hält das Programm schon vor der ersten Ausgabe an und wir laufen per „Step Over“ die nächsten Schritte manuell wie in der folgenden Grafik erkennbar.



The screenshot shows a C code editor window titled "D:\projekte\AD_LCD\main.c". The code is as follows:

```
117      /* Konvertierung des AN0 Eingang */
118      B ADCON0bits.GO = 1; // Konvertierung starten
119      while (ADCON0bits.GO); // Wenn Konvertierung fertig -> ADON0_GO =
120
121      /* 8-Bit Werte auf ein 16-Bit Bereich schreiben */
122      ADCResult.v[0] = ADRESL;
123      ADCResult.v[1] = ADRESH;
124
125      /* 10-Bit Wert in ASCII String konvertieren */
126      itoa(ADCResult.Val, ANOString);
127
128
129      /*
130      * Konvertierung des AN1-Eingang
131      *
132      * Auszug aus MHCP-Stack Source:
133      * Am PICDEM.net Board sollten RA2-RA7 digital sein sonst funkti
134      * LED, LCD und NIC nicht korrekt.
135      * Da es keinen Mode gibt bie dem AN0 und AN1 analoge Inputs und
136      * digitale Pins sind, wir vorübergehend RA3 während der AD-Wand
137      * analog geschaltet. Wenn die AD-Wandlung um ist wird RA2 wiede
138      * auf digital geschaltet.
139      */
140      /* Right justified - von "rechts" nach "links" lesend, RA3 -> a
141      ADCON1 = 0b10000100;
142
```

A red circle with a white 'B' is placed next to line 118, indicating a breakpoint. A green arrow points to line 141, indicating the current execution position.

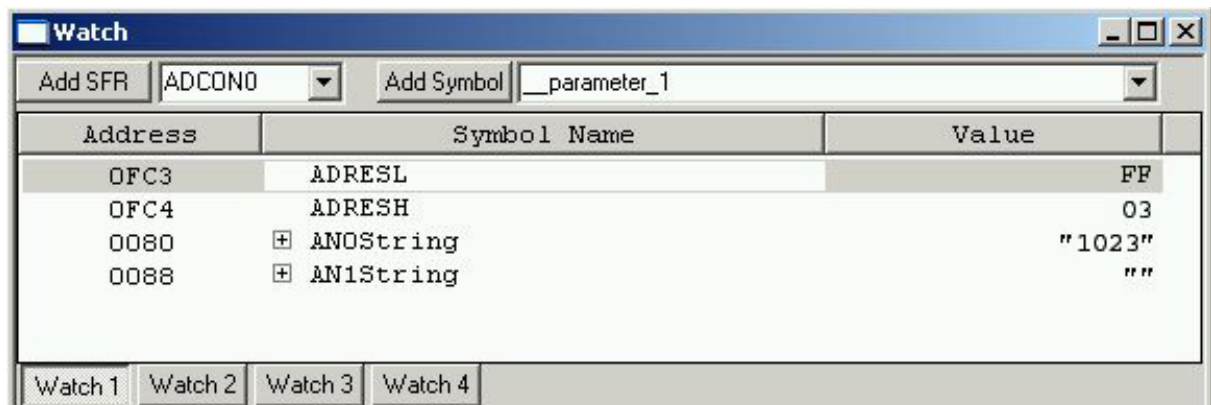
Bild 21 – ICD 2 Debuggen – Breakpoint gesetzt

Folgende Schritte ergeben sich mit dem „Sep Over“ Button:

- Umwandlung starten
- Warten bis fertig
- ADRESL in v[0] schreiben – niedere 8-Bit
- ADRESH in v[1] schreiben – höhere 8-Bit
- 16-Bit Ergebnis in String schreiben

Aktuell steht das Programm bei der Konfiguration des ADCON1 Registers für die Umwandlung des AN1 Eingangs.

Da wir am Anfang unser Watch Window geöffnet und eine Variablen sowie Register eingefügt haben können wir uns diese nun anschauen und werden folgendes sehen können:



The screenshot shows a 'Watch' window with a table of memory addresses and their corresponding values. The table has three columns: Address, Symbol Name, and Value. The rows are as follows:

Address	Symbol Name	Value
0FC3	ADRESL	FF
0FC4	ADRESH	03
0080	⊕ AN0String	"1023"
0088	⊕ AN1String	""

Below the table, there are tabs for Watch 1, Watch 2, Watch 3, and Watch 4. The 'Add SFR' dropdown is set to 'ADCON0' and the 'Add Symbol' dropdown is set to '__parameter_1'.

Folgender Code wurde kurz zuvor ausgeführt:

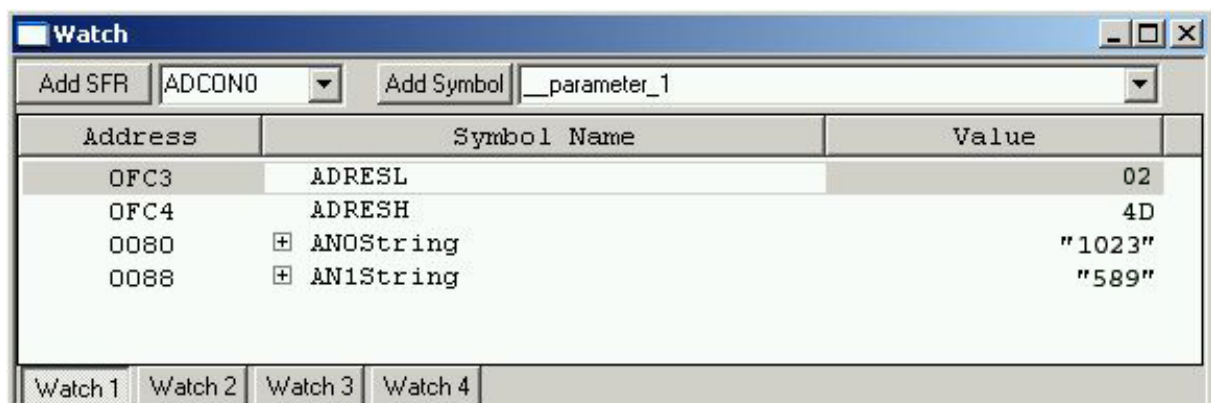
```
/* Konvertierung des AN0 Eingang */
ADCON0bits.GO = 1; // Konvertierung starten
while(ADCON0bits.GO); // Wenn Konvertierung fertig -> ADON0 GO = 0

/* 8-Bit Werte auf ein 16-Bit Bereich schreiben */
ADCResult.v[0] = ADRESL;
ADCResult.v[1] = ADRESH;

/* 10-Bit Wert in ASCII String konvertieren */
itoa(ADCResult.Val, AN0String);
```

Da wir konfiguriert haben, dass das LSB des Umwandlungsergebnis an die Stelle des LSB des Registers ADRESL stehen soll, kann man erkennen, dass in ADRESL 8 niederwertigeren Bits des Ergebnisses stehen. Zudem kann man auch das Ergebnis in dem String lesen, da es im letzten Schritt in den String geschrieben wird.

Wenn wir nun die Schritte bis zum Ende der Funktion weitergehen können wir folgende Watch erkennen:



The screenshot shows the 'Watch' window after the code execution. The table of memory addresses and their corresponding values is updated as follows:

Address	Symbol Name	Value
0FC3	ADRESL	02
0FC4	ADRESH	4D
0080	⊕ AN0String	"1023"
0088	⊕ AN1String	"589"

The tabs and dropdowns remain the same as in the previous screenshot.

Man kann hier sehr gut Erkennen, dass die Special Function Register ADRESL und ADRESH immer nur die Werte der letzten Umwandlung halten. Das bedeutet, wenn wir uns den Wert des AN0 Eingangs nicht in dem String zur Ausgabe gespeichert hätten wäre dieser hier verloren gewesen.

5.3 Zeichen über serielle Schnittstelle und Anzeige auf LCD

In diesem Projekt werden von einem Terminal Programm Zeichen geschrieben und auf dem LCD-Display ausgegeben.

Hier wird ausführlich die Serielle Schnittstelle, Interrupts sowie das LCD-Display beschrieben.

Das Projekt befindet sich im Ordner /Projekte/

5.4 Timer Interrupts und Serielle Schnittstelle Interrupt

Hier werden ausführlich Interrupts beschrieben, was allerdings schon in der Ansteuerung der Peripherie auch noch einmal getan wurde. Somit steht hier auch keine Quelltext sondern nur der Verweis auf dies Projekt im Ordner /Projekte/

6 Projektaufgabe und Durchführung

6.1 Einführung in das Projekt

6.1.1 Aufgabenstellung

Die Aufgabenstellung lautete auf der Homepage von Prof. Dr. Linn wie folgt:

„Programmierung eines Microchip Controllers zur Steuerung eines Gleichstrom Motors mit PIC C18 (übers Netz).“

Die Motivation die uns dazu geführt hat Anhand dieser Aufgabenstellung uns für diese Projekt zu entscheiden war, dass wir ein Objekt, in diesem Fall ein Motor, anhand eines Microcontroller über das Internet steuern sollten. Bisher hatten wir nur mit „nahen“ Objekten zu tun, die also eine direkte Verbindung zu ihrem Controller oder Rechner haben. Es war daher also recht Spannend dies nun über das Internet/Ethernet zu realisieren.

6.1.2 Verwendete Materialien

Für unser Projekt, dem Ansteuern eines Gleichstrommotors, bekamen wir nach und nach folgende Hard- und Software:

- Das PICDEM.net Demo Board
- Dickes Embedded Ethernet Buch
- C18 ANSI C Compiler
- MPLAB IDE v6.10
- ICD 2 In-Circuit Debugger
- Einen Gleichstrommotor aus einem HP-Drucker und den dazugehörigen
- HEDS 9000 Sensor für die Lochscheibe
- IC von Alegro SA3968A – H-Brücke

6.1.3 Verlagerung des Projektschwerpunkts

Wenn man bis hierhin die Dokumentation gelesen hat kann man erkennen, dass die ersten 5 Kapitel eine recht Ausführliche Einführung in die gegebene Soft- und Hardware ergeben. Ungefähr in dieser Reihenfolge und auch in dieser Gewichtung wie in der Dokumentation verlief unser Projekt. Das Bedeutet, dass wir mehr wissenschaftliche Forschung und Untersuchungen an dem PICDEM.net Demo Board sowie deren Peripherie getätigt haben als unser eigentliches Projektziel verfolgen zu können. Das eigentliche Hauptproblem waren die Dokumentationen zu dem PICDEM.net Demo Board sowie für den MPLAB DIE v6.x. Da diese so gut wie nicht vorhanden waren, vor allem die von dem PICDEM.net Demo Board, musste sehr viel Aufwand in die Untersuchung des PICDEM.net Demo Board gesteckt werden. Wenn also eine ausführliche und gute Dokumentation verfügbar gewesen wäre, hätte man eine lange Zeit sparen und sich dem eigentlichen Projekt widmen können.

Somit hat sich im Ganzen der Schwerpunkt des Projekts auf Forschung und Untersuchung verlagert und auch die Dokumentation hat dadurch den Schwerpunkt der ausführlichen Einführung in die Materie.

Dennoch war es unser Anliegen das gegebene Projektziel zu erreichen was uns gegen Ende noch geglückt ist. Daher gibt es die nächsten Kapitel über die Realisierung der Projektaufgabe.

6.2 Realisierung Hardware

6.2.1 Einführung

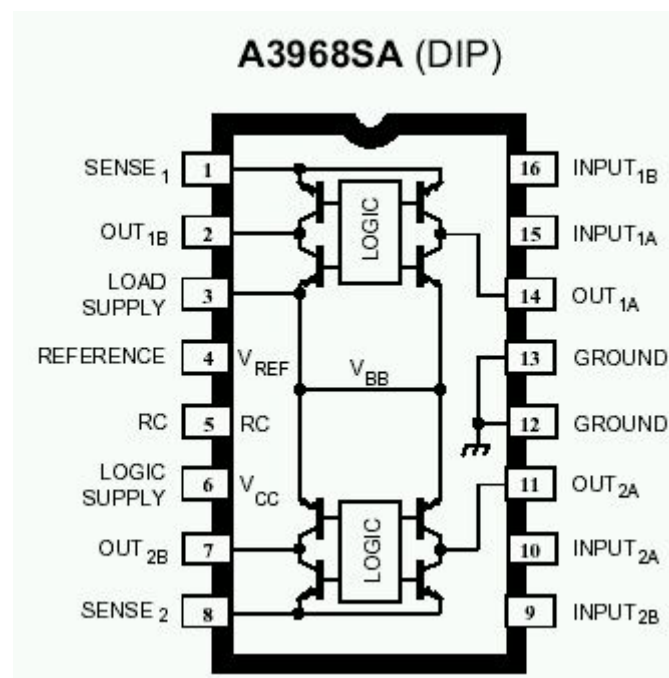
Es gilt einen Motor durch den Microcontroller zu steuern. Wie das im speziellen auf der Softwareseite gelöst wurde wird im nächsten Kapitel beschrieben. Grundsätzlich gilt, dass der Motor über eine H-Brücke mit Pulsweitenmodulation gesteuert werden soll.

Wir haben hierfür einen ausgebauten HP-Druckermotor erhalten, der einen HEDS 9000 Sensor enthält. Zudem bekamen wir eine H-Brücke mit eigener Logik in einem IC, so dass man diese H-Brücke im Bedarfsfall schnell und einfach auswechseln kann.

Da noch niemand mit dieser speziellen Hardware gearbeitet hatte musste diese also erst untersucht und Datenblätter gefunden werden weshalb hier nun auch genauer auf die Hardware eingegangen wird. Zudem galt es seine erworbenen Elektrotechnik Kenntnisse wieder aufzufrischen und anzuwenden.

6.2.2 Allegro SA3968A H-Brücke

Das besondere an diesem IC ist, dass er zwei Motoren gleichzeitig ansteuern kann. Jedes IC beinhaltet zwei H-Brücken welche einen kontinuierlichen Output von $\pm 650\text{mA}$ und eine Arbeitsspannung bis 30 V haben können. Ein Motor kann über eine interne feste Frequenz oder Pulsweitenmodulation gesteuert werden.



Da wir diesen Motor mit PWM steuern wollen muss der RC und der SENCSE Eingang auf MASSE gesetzt werden. Somit ist die interne festen Frequenz-Steuerung ausgeschaltet und jetzt kann man über die INPUT Pins das PWM Signal anlegen.

Die interne Logik arbeitet mit folgender Wahrheitstabelle:

INPUT _A	INPUT _B	OUT _A	OUT _B	Description
L	L	L	L	Brake mode
L	H	L	H	"Forward"
H	L	H	L	"Reverse"
H	H	Z	Z	Disable

Der Zustand Z bedeutet „Hoher Widerstand“.

Es ist also Möglich den Motor auf verschiedenen Arten zu bremsen. Die eine Möglichkeit ist das abrupte sofortige Bremsen (Brake mode) und die andere ist das Abschalten des Motors, was bedeutet, dass dieser in seinen Umdrehungen ausläuft. Wenn also eine etwas größere Trägheit vorhanden ist dreht sich der Motor erst noch ein wenig weiter bis er stehen bleibt.

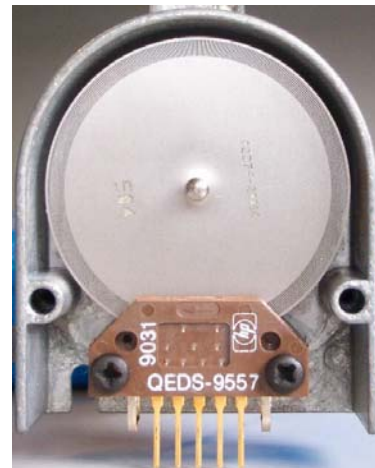
An Pin 3 – dem Load Supply wird die Spannungsversorgung für den Motor angeschlossen. Wenn die H-Brücke nun dem Motor Spannung geben möchte wird dies von der an Pin 3 anliegenden Spannung genommen.

Das genaue Datenblatt zu diesem IC ist im Anhang verfügbar.

6.2.3 Motor

Der Motor ist ein alter HP-Druckermotor der ausgebaut wurde. Das schöne an diesem Motor ist, dass er eine sehr feine Rasterung hat. Er kann also sehr fein angesteuert und zum Drehen gebracht werden.

Zudem hat die Lochscheibe, die für die Erkennung der Drehrichtung sowie Geschwindigkeit, eine hohe Anzahl von Löchern was bedeutet, dass man sehr genaue Ergebnisse bekommt.



Am linken Ende des Motors (linkes Bild) kann man die Drehachse mit seinen Kerben erkennen. An dieser kann man, durch einen Aufsatz ein Objekt, welches gesteuert werden soll, befestigen.

Auf dem rechten Bild kann die Lochscheibe erkennen welche am hinteren Ende des Motors eingebaut ist. Die Lochscheibe hat 512 Löcher welche durch 2 optische Sensoren abgetastet werden. Folgende Grafik beschreibt das Verhalten der Sensoren.

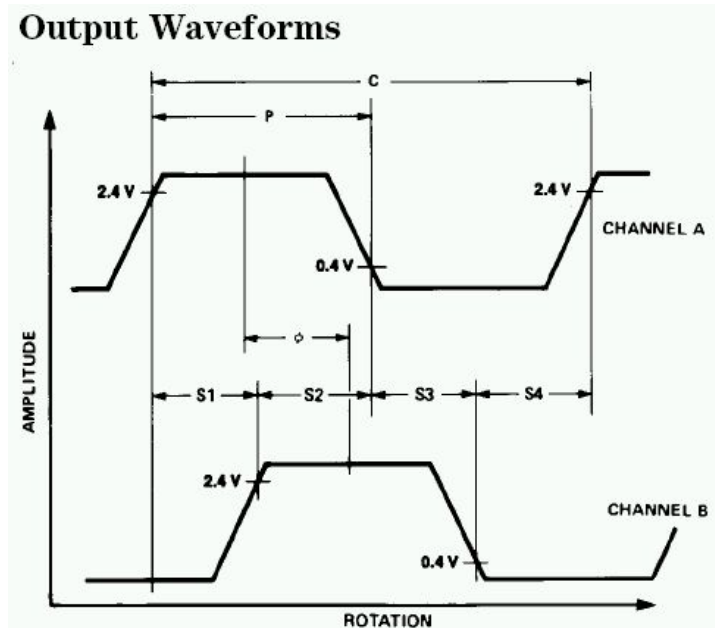
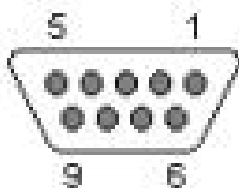


Bild 22 - Motorsensor - Verhalten der optischen Sensoren

Das Prinzip der Drehrichtungserkennung ist die Phasenverschiebung der zwei Amplituden. Je nachdem in welche Richtung sich der Motor dreht ist die zweite Kennlinie 90° vor oder hinter der ersten Kennlinie.

Um die Drehrichtung auswerten zu können muss die beiden Signale an zwei Eingänge des Microcontrollers gelegt werden die jedes Mal bei einer ansteigenden Amplitude einen Interrupt auslösen. Mehr dazu im Kapitel Realisierung Software.

Auf dem linken Bild kann man auch noch den Verbindungsstecker, der den Motor mit dem PICDEM.net Demo Board verbindet, erkennen. Wir haben hier einen RS 232 Stecker verwendet dessen Pinbelegung folgendermaßen ist:



PIN	Bedeutung
2	Amplituden Sensor 2 – Channel B
3	Amplituden Sensor 1 – Channel A
4	Ground
7	+5V Vcc

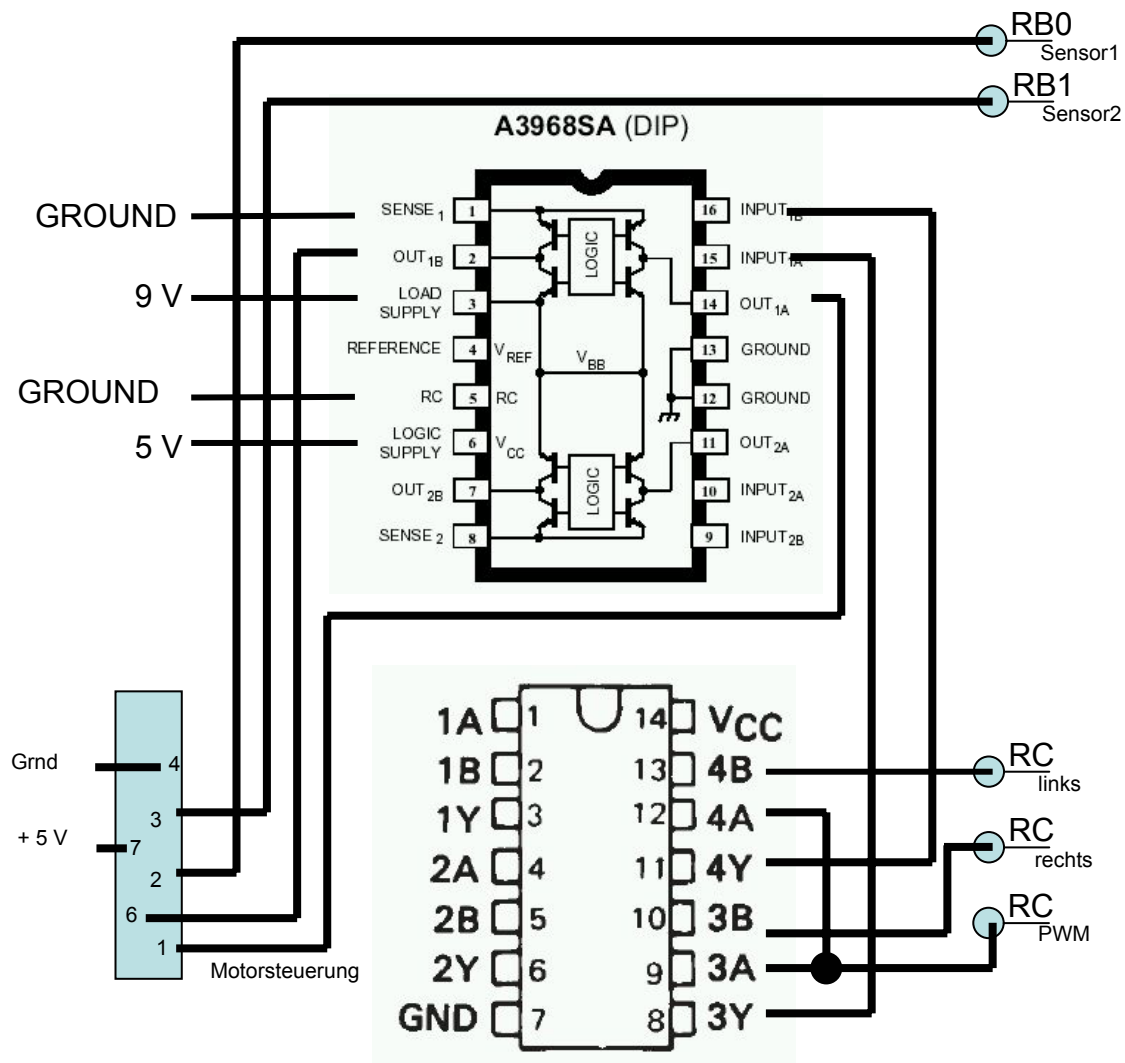
Bei der externen Spannungsversorgung für den Motor auf dem Prototyp Bereich sollte noch erwähnt werden, dass der äußere Kreis die Masse und die Innenseite +9V ist.

6.2.4 Externe Schaltung

Jetzt kommen wir zum Kern der externen Hardware. Es musste sich nämlich eine Lösung für das folgende Problem finden. Wir haben zwei Eingänge bei der H-Bücke aber drei Signale: rechts, links und das PWM-Signal.

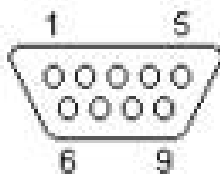
Das Bit für Links liegt an RC0,
das Bit für Rechts liegt an RC1 und
die PWM wird an RC2 ausgehen.

Die Lösung für das Problem ist, dass wir die PWM Signale zusammen mit jeweils Rechts und Links auf eine UND Gatter laufen lassen. Das eine wird dann auf den INPUT1 und das andere auf INPUT2 der H-Brücke gelegt. Wenn der Motor nun nach Rechts laufen soll, wird einfach das Bit an RC1 auf 1 und an RC0 auf 0 gesetzt. Somit schaltet nur das UND Gatter für die Rechts-Stellung durch und gibt das Signal an den entsprechenden INPUT Eingang der H-Brücke weiter. Folgender Schaltplan sollte dies noch mehr verdeutlichen.



Auf der Prototypfläche des PICDEM.net Demo Board wurden somit diese beiden IC's, eine RS 232 Buchse für den Motor und eine zusätzliche Spannungsversorgung für den Motor installiert.

Die RS 232 Buchse hat folgende Belegung:



PIN	Bedeutung
1	Spannungsversorgung Motor Out 1A
2	Amplituden Sensor 2 – Channel B – an RB0
3	Amplituden Sensor 1 – Channel A – an RB1
4	Ground
6	Spannungsversorgung Motor Out 1B

7	+5V Vcc
---	---------

Durch dieses einfache Steckverfahren kann man den Motor oder die Spannungsversorgung nicht verkehrt anschließen. Da hier auch nur IC's verwendet werden ist auch dessen Auswechslung sehr einfach.

6.3 Realisierung Software

6.3.1 Ansteuerung des Motors über die serielle Schnittstelle

Zur Mitte des Projektes war es nicht sicher, ob wir das Ziel der Ansteuerung über Ethernet, erreichen würden. Daher hatten wir als mittelfristiges Ziel den Motor per serielle Schnittstelle anzusteuern. Die Idee war ein einfaches Terminalprogramm, wie Hyperterminal, zu nutzen um Befehle an den Chip zu schicken. Um zu überprüfen, was am Chip ankommt sollte eine Ausgabe auf dem LCD und den UserLEDs erfolgen.

Der Motor selbst wurde indirekt über die H-Brücke mit dem chipinternen PWM-Modul gesteuert welche im Kapitel zuvor erläutert wurde. Um die Drehrichtung zu bestimmen wurden die Pins 0 und 1 des Digitalen I/O-Ports C als Ausgänge konfiguriert.

Das PWM-Modul wurde mit den Bibliotheksfunktionen des C18-Compilers gesteuert.

```
OpenPWM1(0xFF); // Initialisiert PWM mit einer Periode
SetDCPWM1(0x00); // Setzt den DutyCycle des PWM-Moduls
```

Der USART wurde direkt per Registereintrag konfiguriert und auf interruptgesteuerten Empfang eingestellt.

```
void USARTInit(void)
{
    IPR1bits.RCIP    = 1; // Make USART RX Interrupt High Prio

    SPBRG = 15;
    TXSTAbits.BRGH   = 0; // Baudrate Low Speed select
    TXSTAbits.SYNC    = 0; // Asynchroner Modus
    RCSTAbits.SPEN    = 1; // Serial Port Enabled
    PIR1bits.RCIE     = 1; // Enable USART RX Interrupt
    RCSTAbits.RX9     = 0; // 8-Bit Empfang
    RCSTAbits.CREN    = 1; // Enable Receiver
}
```

```
#pragma interrupt USARTrxISR
void USARTrxISR(void)
{
    char data;

    while(PIR1bits.RCIF)
    {
        data = RCREG;
        read_write_buffer(&data, WRITE);
    }
}
```

Zur Zwischenspeicherung der in der ISR empfangenen Werte dient ein Ringpuffer, der in einer Funktion mit statischen Array als Datenbehälter und jeweils einem statischen Lese und Schreibzeiger realisiert ist.

```
unsigned char read_write_buffer(char *c, unsigned char rw)
```

```

{
    volatile static char buffer[BUFF_SIZE]; // Statischer Puffer
    volatile static unsigned read_ptr = 0;
    volatile static unsigned write_ptr = 0;

    MASK_USART_RX_INT;

    if(rw == READ)
    {
        if(read_ptr != write_ptr) // Puffer enthält Zeichen
        {
            *c = buffer[read_ptr++];
            if(read_ptr >= BUFF_SIZE)
                read_ptr = 0;
            UMASK_USART_RX_INT;
            return TRUE;
        }
        else // Puffer leer
        {
            UMASK_USART_RX_INT;
            return FALSE;
        }
    }
    else // rw == WRITE
    {
        buffer[write_ptr++] = *c;
        if(write_ptr >= BUFF_SIZE)
            write_ptr = 0;
        UMASK_USART_RX_INT;
        return TRUE;
    }
}

```

Die benötigten Routinen zur Ansteuerung des LCD wurden der modifizierten Bibliothek des Microchip TCP/IP-Stack Beispielsprogramm entnommen wie sie schon unter 4.3.3 beschrieben wurde.

Die Befehlsstruktur war nun sehr einfach gehalten, weil es ja nicht die endgültige Lösung sein sollte. Um die Motorleistung zu verändern muss man nur per Hyperterminal eine Zahl aus dem Bereich 0 – 1023 eintippen. Diese Zahl wird auf dem LCD ausgegeben und kann auch korrigiert werden. Bestätigt man mit Enter, so wird der String in einen unsigned Wert konvertiert und als neuer DutyCycle verwendet.

Mit l und r kann man die Drehrichtung ändern und mit b und d kann man den Motor bremsen bzw. ausschalten.

Als die externe Schaltung nun funktionstüchtig war konnte man die Decoderleitungen des Motors, die an die Pins 0 und 1 des Digitalen I/O-Ports B angeschlossen wurden, auswerten und zum Ermitteln von Drehgeschwindigkeit und DrehRichtung nutzen. Pin 0 und 1 der Ports B sind jeweils externe Interrupt Leitungen, die separat konfiguriert werden können.

Zunächst reichte es nur Pin 0 als Interruptleitung zu benutzen. Die ISR der Externen Interrupts 0 konnte schon zur Ermittlung der Drehrichtung genutzt werden und um einen Zähler zu inkrementieren.

```

#pragma interrupt DecoderISR
void DecoderISR(void)
{
    direction = PORTBbits.RB1; // Wenn Pin 1 = 0 -> Linksdrehung
    tick_counter++;
    INTCONbits.INT0IF = 0; // Löschen des Interrupt Flags
}

```

Um jetzt die Drehgeschwindigkeit zu bestimmen mussten wir einen Timer initialisieren, der nach jedem Überlauf einen Interrupt auslöst. Dessen ISR liest den Wert des Zählers und verrechnet ihn mit einer Konstanten, um die U/min zu bestimmen. Ist die Drehung des Motors zu diesem Zeitpunkt eine Rechtsdrehung, so wird dem Wert ein '-' vorangestellt. Danach wird der Zähler wieder auf 0 zurückgesetzt.

```
// Zeitkonstante zur Berechnung der Drehgeschwindigkeit in U/min
// T = 0.0000002[s](Tosc) * 16[ ](TMR0Prsc) * 65536[ ](16-Bit-Timer)
// 512 [ticks/U]
// TMR0Period = (T[s]/60[s/min]) * 1024[ticks/U]
const double TMR0Period = 1.7895697066666672; // [ticks*min/U]
```

```
#pragma interrupt Timer0ISR
void Timer0ISR(void)
{
    speed = (long)((double)tick_counter/TMR0Period); // [ticks]/[ticks*min/U] = U/min
    if(direction == RIGHT SPIN)
        speed = -speed;
    tick_counter = 0;
    INTCONbits.TMR0IF = 0;
}
```

Somit ließ sich nun die Drehgeschwindigkeit recht genau ermitteln und auf dem LCD-Display auszugeben.

Immerhin ist es nun Möglich, trotz aller Dokumentations- und Anfangsschwierigkeiten den Motor, im Zusammenspiel mit Microcontroller und einer externen Schaltung, über die serielle Schnittstelle zu steuern.

Um eine Verbindung mit dem Hyperterminal zu dem Microcontroller herzustellen gelten die gleichen Einstellungen für das Hyperterminal wie im Kapitel 1 – Getting Started beschrieben.

Dieser Stand wurde auch in der Vorlesung präsentiert und die Ansteuerung per Ethernet war in Arbeit.

6.3.2 Motoransteuerung per Ethernet

Als nächstes stand auf dem Plan dem Motor über die Ethernetschnittstelle zu steuern. Auf diesem Punkt sollte eigentlich unser Schwerpunkt liegen. Durch die schon oft in der Dokumentation beschriebenen Schwierigkeiten sind wir leider erst kurz vor dem Ende unserer Verfügbaren Zeit an diesen Punkt gekommen. Da aber unser Ehrgeiz ausreichte um wenigsten den Versuch zu starten doch noch das gesamte Projektziel zu erreichen wurde quasi Tag und Nacht an dieser Lösung hier gearbeitet.

Grundsätzlich sollte man an dieser Stelle auch auf dem mitgelieferten MCHPStack hinweisen. Da wir nur noch sehr wenig Zeit hatten ersparten wir uns die Mühe einen eigenen TCP Stack zu bauen und haben daher als Grundlage den MCHPStack verwendet.

In dem Source-Code des MCHPStacks haben wir alle für uns unnötigen Module, wie z.B. DHCP, HTTP oder FTP rausgeschmissen und unseren UDP Client reinprogrammiert.

Im Quellcode, den Header Files, kann man auch immer die Änderungen sehen, da wir genau dokumentiert haben was wir geändert haben.

Wenn man nun dieses Projekt an dieser Stelle hier starten würde, oder neu vergeben, dann könnte man den Stack, das Protokoll und die gesamte Anwendung über Ethernet bestimmt perfektionieren. Aber das war an dieser Stelle nicht mehr

unser Gedanke, vielmehr wollten wir die Anwendung überhaupt noch über Ethernet zum laufen bringen.

Daher geht die Dokumentation auch nicht in perfektionistischer Weise an die Realisierung sondern es soll ein Überblick geschaffen werden wie wir vorgegangen und es realisiert haben.

Vorgehensweise:

Da wir die Ansteuerung des Motors schon über die serielle Schnittstelle gelöst haben musste dieser durch den Chip auch nichts geändert werden. Und auch die Ermittlung der Geschwindigkeit und Drehrichtung konnte beibehalten werden. Das Einzige was sich in der Hinsicht geändert hat, dass jetzt auch Pin 1 an Port B einen Interrupt auslöst, was zunächst nur eine höhere Genauigkeit bedeutet.

Wir kamen zu dem Schluss, UDP als verbindungsloses Protokoll, die ideale Methode für die Datenübermittlung an den Chip ist. Doch die Frage war: „Was wollen wir übertragen?“. Als erstes fiel uns die XDATA Struktur aus PDV ein, doch die war dafür konzipiert einen Remotezugriff auf dem UART eines Hosts zu realisieren, was hier etwas umständlich wäre, da man ja direkt auf die Register des PWM und PORTC zugreifen könnte. Allerdings wäre es praktisch die XDATA Struktur weiterhin benutzen zu können, um rückwärtskompatibel zu sein.

Also behielten wir die XDATA Struktur fügten noch drei Strukturen hinzu:

- CDATA - Für den direkte Übertragung aller relevanten Werte
- SDATA - Zur Übermittlung einer Sollgeschwindigkeit
- EDATA - Zur Übermittlung von Fehlermessages an einen Client

Nun hatten wir das Problem, dass wir beim Empfang eines Pakets nicht wissen konnten, um was für eine Struktur es sich handelt. Und ein Byte vorneweg senden konnten wir nicht, wenn wir rückwärts kompatibel zu alten Programmen bleiben wollten. Da allerdings der erste Wert der XDATA Struktur 16-Bit breit ist, aber nur die niederwertigsten 2 Bit nutzen, sind nun die höchstwertigen 3 Bit als Identifikation reserviert. Und damit wir an XDATA nichts ändern müssen sind diese drei Bit bei der Struktur 000.

Im folgenden Listing kann man die verwendete CDATA und die anderen beiden Strukturen erkennen.

```
/* Aus PDV bekannte XDATA Struktur um einen Remotezugriff auf
   eine (virtuelle) RS232-Schnittstelle zu realisieren.
   Wird aus Kompatibilitaetsgruenden zu alten Aufgaben implementiert
   Siehe ProcessXDATA() in motor server.c fuer Details
*/
typedef struct XDATA
{
    WORD rdwr;
    WORD port;
    WORD value;
    DWORD t0;
    WORD status;
}XDATA;

/* Control Data Struktur erlaubt Remotezugriff auf die Register
   die PWM-relevanten Register des PIC18F452
   Die ersten zwei Bit des von direction müssen 1 sein, um die
   CDATA-Struct als solche zu kennzeichnen.
   direction kann folgende Werte annehmen:
       0x0000 -> Break
       0x0001 -> Rechtsdrehung
       0x0002 -> Linksdrehung
       0x0003 -> Disable
```

```
    dutycycle hat einen Wertebereich von
        0x000 - 0x3FF
    in speed-val wird die aktuelle Geschwindigkeit in U/min
    zurückgeliefert
*/
typedef struct _CDATA
{
    WORD direction;
    WORD dutycycle;
    long speed val;
}CDATA;

/* Speed Data Struktur erlaubt es eine Geschwindigkeit
in U/min anzugeben, die der Motor erreichen soll.
dummy wird nur genutzt um die Struktur als SDATA
Strukt zu kennzeichnen: Bit 12-15 -> 1.
speed_val enthält die Sollgeschwindigkeit
und wird als Antwort die Ist-Geschwindigkeit
zugewiesen
*/
typedef struct SDATA
{
    WORD dummy;
    long speed val;
}SDATA;

/* Error Data Struktur wird als Antwort bei aufgetretenen
Fehlern an den Client gesendet.
dummy dient wieder nur der Kennzeichnung: Bit 15 -> 1.
message enthält de Fehlertext
*/
typedef struct EDATA
{
    WORD dummy;
    char message[32];
}EDATA;
```

Wenn nun ein Paket empfangen wurde, wird dieses erst einmal in einen neutralen Puffer geladen und anschließend überprüft um welche Struktur es sich handelt.

```
// Ausschnitt aus der Funktion MotorServer.c

dataid.Val = (WORD) (*buffer);

if((dataid.v[0]&0xE0) == 0xE0)
    error = ProcessSDATA((SDATA*)buffer);
else if((dataid.v[0]&0xE0) == 0xC0)
    error = ProcessCDATA((CDATA*)buffer);
else if(dataid.v[0]&0xE0 == 0x00)
    error = ProcessXDATA((XDATA*)buffer);
else
    error = EINVALSTCT;
```

Anschließend wird die erkannte CDATA Struktur zur Auswertung der Daten weitergereicht. Es beinhaltet den Dutycycle und die Drehrichtung.

Der Motor wird nun wieder wie bei der seriellen Ansteuerung gesteuert und gibt die gleichen lokalen Ausgaben. Wir haben hier aber zusätzlich noch einen Rücksende Verfahren eingebaut.

Es wird also wieder ein CDATA mit der Information „Geschwindigkeit“ zurück an den Client gesendet. Daher ist es Möglich die gesamte Kontrolle vom Client aus zu haben.

Zusätzlich wurden Fehlermeldungen definiert die, wenn sie auftreten, an den Client zurückgesendet und dort ausgegeben werden.

Hier sollte noch erwähnt sein, dass nur die CDATA Struktur ausführlich getestet wurde. Theoretisch müssten die beiden anderen Strukturen auch funktionieren, jedoch wurden sie nicht in der Praxis getestet.

Die folgende Grafik zeigt das User-Interface mit welchem man den Motor steuern kann. Es ist möglich periodisch zu senden, wenn man das entsprechende Häkchen in der Checkbox setzt.

Zudem werden alle Informationen die Möglich sind (Sendebericht, Fehlermeldungen) in der Listbox ausgegeben:



Der Sourcecode mit der gesamten Projektumgebung befindet sich auf der CD im Ordner /Projekte/Motorsteuerung via Ethernet/

Somit können Erweiterungen oder Änderungen Problemlos eingefügt und der Microcontroller neu programmiert werden.

6.4 Fazit

Wir wollen hier nur noch ein kurzes Fazit über das Projekt geben. Insgesamt sind wir doch sehr glücklich darüber, dass wir es, vielleicht nicht in Perfektion, geschafft haben den Motor via Ethernet zu steuern.

Zudem haben wir sehr viel im Bereich Microcontroller und der Entwicklung von Anwendungen in diesem Bereich gelernt. Allerdings haben wir auch Bemerkt, wie wichtig doch Dokumentationen sind und wie man doch ohne diese Aufgeschmissen ist.

Dennoch war das gesamte Projekt eine sehr gute Erfahrung und hatte viele interessante Themen zu bieten.

Anhang

6.5 Datenblätter/Quellen

Wenn spezielle Informationen oder Daten angegeben worden sind, dann kann man sie in den folgenden Datenblätter oder URLs nachlesen und finden:

- PIC18Fxx2 Data Sheet.pdf
- MPLAB.net Demonstartion Kit Errata Sheet.pdf
- MPLAB.net InternetEthernet Demo Board Internet Solutions.pdf
- MPLAB IDE v6.xx Quick Start Guide.pdf
- MPLAB ICD 2 User Guide.pdf
- MPLAB-C18-Getting-Started.pdf
- MPLAB-C18-Libraries.pdf
- MPLAB-C18-Users-Guide.pdf
- MPLAB-CXX Compiler User's Guide .pdf
- PIC18Fxxx_comprehensive_tutorial_containing_7Mb_of_info.pdf
- HITACHI LCD-Display HD44780 - Data Sheet.pdf
- IC - ALEGRO SA3968A - HBruecke.pdf
- IC - UND-Gatter 74LS08.pdf

Alle Datenblätter sind auf der CD im Ordner /pdf/ zu finden.

Zusätzlich wurde als Quelle das Internet mit folgenden URLs benutzt:

www.microchip.com

www.mircochipc.com

www.microchip.com/1010/pline/tools/picmicro/devenv/mplabi/mplab6/index.htm/

www.microchip.com/1010/pline/tools/picmicro/code/mplab18/

www.microchip.com/1010/pline/tools/picmicro/icds/icd2/

www.google.de

6.6 Ordnerstruktur der CD

Ornder	Inhalt
Bilder	Enthält alle in der Dokumentation verwendeten Bilder.
C18 Compiler	Installationsdateien des C18 Compiler.
MCHPStack	Die Mitgelieferte Firmware als Source-Code zum entpacken und weiterarbeiten.
MPLAB IDE	Die Installationsdatei der Entwicklungsumgebung von MPLAB Version 6.20.und die eben von Microchip online gestellte Version 6.30
pdf	Alle Data-Sheets, Dokumentationen und Quellen in PDF Form die wir im Laufe unseres Projekts gefunden und verwendet haben.
Präsentation	Unsere Powerpoint Präsentation so wie wir sie Präsentiert haben sowie die verwendeten Bilder in der Präsentation.
Projekte	Alle Unterprojekte die wir im Laufe der Vertiefung erarbeitet haben. Eins davon wird im Kapitel Tutorial verwendet, die anderen können zum Selbststudium genutzt werden, da alles ausführlich

	kommentiert ist. Die beiden Hauptprojekte sind auch in diesem Order in den jeweils gekennzeichneten Unterordnern.
UDP Client	Der genutzte UDP Client mit Source Code und evtl. gebraucht dlls für ein System auf dem keine Visual Studio installiert ist. Dazu einfach die dlls in das Verzeichnis kopieren in dem die ausführbare Datei liegt.

6.7 Source Code motor_udp

6.7.1 motor_server.h

```

/*****
 *
 *      Prozessdatenverarbeitung Vertiefung
 *      Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 *      Demobords und dem Microchip TCP/IP-Stack
 *
 *****/

 *
 *      Markus Kühle
 *      Marco Kosinski      534931
 *
 *****/
 *
 *      motor_server.h
 *****/

#ifndef _MOTOR_SERVER_H_
#define _MOTOR_SERVER_H_

/*****
 *
 *      Headerdateien
 *****/

#include <pl8cxxx.h>
#include "compiler.h"
#include "stacktsk.h"

/*****
 *
 *      Definitionen
 *****/

#define CHAR_MAX 16

// Defines fuer Simulation der seriellen Schnittstelle
#define DPORT 0xF8 /* Datenport an RS232 */
#define LCR 0xFB /* LineControlRegister */
#define MCR 0xFC /* ModemControlRegister */
#define MSR 0xFE /* ModemStatusRegister */

// Masken fuer MSR
#define DCTS 0x01 // Aenderung an CTS
#define DDSR 0x02 // Aenderung an DSR
#define CTS 0x10 // Linker Sensor
#define DSR 0x20 // Rechter Sensor

// Masken fuer MCR
#define DTR 0x01 // Richtungsschalter FORWARD -> 1

// Masken fuer LCR
#define BRKST 0x40

#define RONLY 1
#define WRONLY 2
#define RDWR 3

// Errorcodes

#define NOERROR 0
#define EINVALSTCT 1
#define EINVALDC 2
#define ENORDONLY 3
#define ENOWRONLY 4
#define ENORDWR 5
#define EINVALRDWR 6
#define EXPWM 7

/*****
 *
 *      Typdefinitionen
 *****/

/*
 *      Aus PDV bekannte XDATA Struktur um einen Remotezugriff auf
 *      eine (virtuelle) RS232-Schnittstelle zu realisieren.
 *      Wird aus Kompatibilitaetsgruenden zu alten Aufgaben implementiert
 *      Siehe ProcessXDATA() in motor_server.c fuer Details
 */
typedef struct _XDATA
{
    WORD rdwr;

```

```

        WORD port;
        WORD value;
        DWORD t0;
        WORD status;
}XDATA;

/* Control Data Struktur erlaubt Remotezugriff auf die Register
   die PWM-relevanten Register des PIC18F452
   Die ersten zwei Bit des von direction müssen 1 sein, um die
   CData-Strukt als solche zu kennzeichnen.
   direction kann folgende Werte annehmen:
       0x0000 -> Break
       0x0001 -> Rechtsdrehung
       0x0002 -> Linksdrehung
       0x0003 -> Disable
   dutycycle hat einen Wertebereich von
       0x000 - 0x3FF
   in speed-val wird die aktuelle Geschwindigkeit in U/min
   zurückgeliefert
*/
typedef struct _CData
{
        WORD direction;
        WORD dutycycle;
        long speed_val;
}CData;

/* Speed Data Struktur erlaubt es eine Geschwindigkeit
   in U/min anzugeben, die der Motor erreichen soll.
   dummy wird nur genutzt um die Struktur als SData
   Strukt zu kennzeichnen: Bit 12-15 -> 1.
   speed_val enthält die Sollgeschwindigkeit
   und wird als Antwort die Ist-Geschwindigkeit
   zugewiesen
*/
typedef struct _SData
{
        WORD dummy;
        long speed_val;
}SData;

/* Error Data Struktur wird als Antwort bei aufgetretenen
   Fehlern an den Client gesendet.
   dummy dient wieder nur der Kennzeichnung: Bit 15 -> 1.
   message enthält de Fehlertext
*/
typedef struct _EData
{
        WORD dummy;
        char message[32];
}EData;

/*****
 *
 *      Externe Globale Variablen
 *
 *****/

extern BYTE cts;
extern BYTE dsr;
extern BYTE dcts;
extern BYTE ddsr;

/*****
 *
 *      Funktionsdeklarationen
 *
 *****/

void MotorServer(void);
void MotorServerInit(void);
void MotorServerShutdown(void);

BYTE ProcessXData(XData*);
BYTE ProcessCData(CData*);
BYTE ProcessSData(SData*);

unsigned GetDutyCycle(void);
void SetDutyCycle(unsigned);

#endif /* _MOTOR_SERVER_H_ */

```

6.7.2 motor_server.c

```

/*****
 *
 *      Prozessdatenverarbeitung Vertiefung
 *      Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 *      Demoboards und dem Microchip TCP/IP-Stack
 *
 *****/
 *
 *      Markus Kühle
 *      Marco Kosinski      534931
 *
 *****/
 *
 *      motor_server.c
 *
 *****/

/*****
 *
 *      Headerdateien
 *
 *****/

#include <pwm.h>
#include <timers.h>
#include <stddef.h>
#include <string.h>
#include "motor_server.h"

```

```

#include "udp.h"
#include "mac.h"
#include "xlcd.h"
#include "interrupt.h"
#include "decoder.h"
#include "helpers.h"

/*****
 *      Globale Variable
 *****/

BOOL approx = FALSE;
long speed_soll = 0;
//      1234567890123456
ROM char blankLine[] = "          ";

ROM char *errmsg[] = { "",
                      "Invalid Structure was sent",
                      "DutyCycle was > 1023",
                      "Port in XDATA was no RDONLY Port",
                      "Port in XDATA was no WRONLY Port",
                      "Port in XDATA was no RDWR Port",
                      "rdwr in XDATA contained Ivalid value",
                      "PWM value was Invalid" };

BYTE cts;
BYTE dsr;
BYTE dcts;
BYTE ddsr;

/*****
 *      Funktionsdefinitionen
 *****/

/* Motorspezifische Initialisierungen */
void MotorServerInit(void)
{
    //Erste Zeile des Displays loeschen
    XLCDGoto(0,0);
    XLCDPutROMString(blankLine);

    INTCONbits.GIEH = 0; // Disable High Prio Interrupts
    INTCONbits.GIEL = 0; // Disable Low Prio Interrupts
    RCONbits.IPEN = 1; // Enable Priority Interrupts

    DecoderInit(); // Decoder Modul Initialisieren

    // Timer und PWM
    OpenTimer0(TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_16);
    OpenTimer2(TIMER_INT_OFF & T2_PS_1_16 & T2_POST_1_1); // PWM Prescaler
    OpenPWM1(0xFF);
    SetDCPWM1(0x00);

    // Daten der virtuellen RS232 Schnittstelle initialisieren
    cts = 32; // 90° Phasenverschoben zu dsr
    dsr = 0;
    ddsr = 0;
    dcts = 0;

    //Ausgangspins konfigurieren
    TRISCbits.TRISC0 = 0; // Laufrichtung Rechts
    TRISCbits.TRISC1 = 0; // Laufrichtung Links
    TRISCbits.TRISC2 = 0; // PWM Ausgang
    TRISAbits.TRISA2 = 0; // LED Links
    TRISAbits.TRISA3 = 0; // LED Rechts

    // Laufrichtung Links vorkonfigurieren
    LATCbits.LATC0 = 0;
    LATCbits.LATC1 = 1;
    LATAbits.LATA2 = 0; // Leuchtet bei 0
    LATAbits.LATA3 = 1; // Leuchtet bei 0

    // Enable all Interrupts
    INTCONbits.GIEH = 1;
    INTCONbits.GIEL = 1;
}

void MotorServerShutdown(void)
{
    ClosePWM1();
    CloseTimer0();
    CloseTimer2();
}

/* Funktion ueberprueft, ob UDP-Packet empfangen wurde,
   checkt, welches Packet empfangen wurde und gibt die
   Daten an die entsprechende Funktion weiter*/
void MotorServer(void)
{
    static UDP_SOCKET ListenSoc = INVALID_UDP_SOCKET;
    const UDP_PORT ListenPort = 4000;

    EDATA *errorframe;

    BYTE buffer[CHAR_MAX];
    BYTE in_pointer = 0;
    BYTE out_pointer = 0;
    WORD_VAL dataid;

    BYTE error = NOERROR;

    if(ListenSoc == INVALID_UDP_SOCKET)
        ListenSoc = UDPOpen(ListenPort, NULL, 0);
    // Falls UDP-Packet empfangen wurde
    if(UDPIsGetReady(ListenSoc))

```

```
{
    while(UDPGet(&buffer[in_pointer++]))
    {
        if(in_pointer == CHAR_MAX)
        {
            UDPPdiscard();
            break;
        }
    }

    /*Die drei MSBs, der ersten 16 Bit werden benoetigt, um
    herauszufinden, um welche Struktur es sich handelt:
    000 -> XDATA
    100 -> EDATA (wird generell nur als Reply gesendet)
    110 -> CDATA
    111 -> SDATA
    Danach die Daten an die jeweilige Funktion weitergeben
    */
    dataid.Val = (WORD)(*buffer);

    if((dataid.v[0]&0xE0) == 0xE0)
        error = ProcessSDATA((SDATA*)buffer);
    else if((dataid.v[0]&0xE0) == 0xC0)
        error = ProcessCDATA((CDATA*)buffer);
    else if(dataid.v[0]&0xE0 == 0x00)
        error = ProcessXDATA((XDATA*)buffer);
    else
        error = EINVALSTCT;

    //Teste ob Socket breit ist zum Senden
    while(UDPIsPutReady(ListenSoc) == FALSE)
        ;
    //Falls Fehler aufgetreten -> EDATA schreiben
    if(error)
    {
        errorframe = (EDATA*)buffer;
        errorframe->dumy = 0x8000;
        errorframe->dumy = swaps(errorframe->dumy);
        strcpypgm2ram((char*)errorframe->message, errmsg[error]);
        in_pointer = sizeof(EDATA);
    }

    // Daten in Transmittpuffer schreiben...
    while(UDPput(buffer[out_pointer++]))
        if(out_pointer == in_pointer)
            break;

    // ..und abschicken
    UDPFlush();

    // Neuen Socket oeffnen
    UDPClose(ListenSoc);
    ListenSoc = UDPOpen(ListenPort, NULL, 0);
}

/* Falls Annaeherung an Sollwert erwuenscht definiert
Anpassung der Motorleistung vornehmen */
if(approx == TRUE && speed_ist < speed_soll)
    SetDutyCycle(GetDutyCycle() + 1);
else if(approx == TRUE && speed_ist > speed_soll)
    SetDutyCycle(GetDutyCycle() - 1);
else
    approx = FALSE; // Sollwert erreicht oder approx = FALSE

// Puffer loeschen und zur Anzeige der Geschwindigkeit
// auf dem LCD nutzen
memset((void*)buffer, 0, CHAR_MAX);
ltoa(speed_ist, (char*)buffer);
strcpypgm2ram((char*)buffer, (ROM char*)"U/min      ");
XLCDGoto(0,0);
XLCDPutString((char*)buffer);
}

//////////Muss noch verifiziert und dokumentiert werden!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

BYTE ProcessXDATA(XDATA *data)
{
    if((data->rdwr & 0xFF) == RDONLY)
    {
        if((data->port & 0xFF) == MSR)
        {
            DISABLE_INTERRUPTS();
            data->value = (dsr << 5)+(cts << 4)+(ddsr << 1)+dcts;
            dcts = 0;
            dsr = 0;
            ENABLE_INTERRUPTS();
        }
        else if((data->port & 0xFF) == DPORT)
        {
            WORD dc;
            dc = (CCPR1L << 2) + ((CCP1CON & 0x30) >> 4);
            if(dc < 205)
                data->value = 0xFF;
            else if(dc >= 205 && dc < 307) // 20% - 30%
                data->value = 0x7F;
            else if(dc >= 307 && dc < 410) // 30% - 40%
                data->value = 0x3F;
            else if(dc >= 410 && dc < 512) // 40% - 50%
                data->value = 0x1F;
            else if(dc >= 512 && dc < 614) // 50% - 60%
                data->value = 0x0F;
            else if(dc >= 614 && dc < 717) // 60% - 70%
                data->value = 0x07;
            else if(dc >= 717 && dc < 819) // 70% - 80%
                data->value = 0x03;
            // 0% - 20%
```

```

        else if(dc >= 819 && dc < 922) // 80% - 90%
            data->value = 0x01;
        else // 90% - 100%
            data->value = 0x00;
    }
    else
        return ENORDONLY;
}
else if((data->rdwr & 0xFF) == WRONLY)
{
    if(data->port == LCR)
    {
        if(data->value & BRKST)
            SetDCPWM1(0x00);
    }
    else if((data->port & 0xFF) == DPORT)
    {
        if(data->value == 0x00)
            SetDCPWM1(922); // 90%
        else if(data->value == 0x01)
            SetDCPWM1(819); // 80%
        else if(data->value == 0x03)
            SetDCPWM1(717); // 70%
        else if(data->value == 0x07)
            SetDCPWM1(614); // 60%
        else if(data->value == 0x0F)
            SetDCPWM1(512); // 50%
        else if(data->value == 0x1F)
            SetDCPWM1(410); // 40%
        else if(data->value == 0x3F)
            SetDCPWM1(307); // 30%
        else if(data->value == 0x7F)
            SetDCPWM1(205); // 20%
        else if(data->value == 0xFF)
            SetDCPWM1(102); // 10%
        else
            return EXPWM;
    }
    else if((data->port & 0xFF) == MCR)
    {
        if(data->value & DSR) // Laufrichtung Links
        {
            LATCbits.LATC0 = 0;
            LATCbits.LATC1 = 1;
            LATAbits.LATA2 = 0;
            LATAbits.LATA3 = 1;
        }
        else
        {
            LATCbits.LATC0 = 1;
            LATCbits.LATC1 = 0;
            LATAbits.LATA2 = 1;
            LATAbits.LATA3 = 0;
        }
    }
    else
        return ENOWRONLY;
}
else if((data->rdwr & 0xFF) == RDWR)
{
    if((data->port & 0xFF) == DPORT)
    {
        WORD dc;
        dc = (CCPR1L << 2) + ((CCP1CON & 0x30) >> 4);

        if(data->value == 0x00)
            SetDCPWM1(922); // 90%
        else if(data->value == 0x01)
            SetDCPWM1(819); // 80%
        else if(data->value == 0x03)
            SetDCPWM1(717); // 70%
        else if(data->value == 0x07)
            SetDCPWM1(614); // 60%
        else if(data->value == 0x0F)
            SetDCPWM1(512); // 50%
        else if(data->value == 0x1F)
            SetDCPWM1(410); // 40%
        else if(data->value == 0x3F)
            SetDCPWM1(307); // 30%
        else if(data->value == 0x7F)
            SetDCPWM1(205); // 20%
        else if(data->value == 0xFF)
            SetDCPWM1(102); // 10%
        else
            return EXPWM;

        if(dc < 205) // 0% - 20%
            data->value = 0xFF;
        else if(dc >= 205 && dc < 307) // 20% - 30%
            data->value = 0x7F;
        else if(dc >= 307 && dc < 410) // 30% - 40%
            data->value = 0x3F;
        else if(dc >= 410 && dc < 512) // 40% - 50%
            data->value = 0x1F;
        else if(dc >= 512 && dc < 614) // 50% - 60%
            data->value = 0x0F;
        else if(dc >= 614 && dc < 717) // 60% - 70%
            data->value = 0x07;
        else if(dc >= 717 && dc < 819) // 70% - 80%
            data->value = 0x03;
        else if(dc >= 819 && dc < 922) // 80% - 90%
            data->value = 0x01;
        else // 90% - 100%
            data->value = 0x00;
    }
}

```

```

        }
        else
            return ENORDWR;
    }
    else
        return EINVALRDWR;

    return NOERROR;
}

// Die Funktion bearbeitet die Daten der CData struct
BYTE ProcessCData(CData *data)
{
    // NetworkByteOrder -> HostByteOrder
    data->direction = swaps(data->direction);
    data->dutycycle = swaps(data->dutycycle);

    if(data->dutycycle > 1023)
        return EINVALDC;

    // Richtungspins und UserLEDs beschalten
    LATC = (PORTC & 0xFC) | (data->direction & 0x03);
    LATA = (PORTA & 0xF3) | ((data->direction & 0x03) << 2);

    // DutyCycle festlegen
    SetDCPWM1(data->dutycycle);

    // Speichern des aktuellen Geschwindigkeitswert in der Struct
    data->speed_val = speed_ist;

    // HostByteOrder -> NetworkByteOrder
    data->direction = swaps(data->direction);
    data->dutycycle = swaps(data->dutycycle);
    data->speed_val = swapl(data->speed_val);

    // Annäherung an Sollwert ausschalten
    approx = FALSE;

    return NOERROR;
}

BYTE ProcessSData(SData *data)
{
    // NetworkByteOrder -> HostByteOrder
    data->speed_val = swapl(data->speed_val);

    speed_soll = data->speed_val;

    // Aktuelle Geschwindigkeit in NetworkByteOrder in Struct speichern
    data->speed_val = swapl(speed_ist);

    // Annäherung an Sollwert einschalten
    approx = TRUE;

    return NOERROR;
}

/* Die Funktion gibt einen modifizierten DutyCycle zurück
   Original DutyCycle: 10Bit 0x000 - 0x3FF
   Modifiziert       : 11Bit 0x000 - 0x7FF
                                0x000 - 0x3FF -> 0x3FF - 0x000 rechtsdrehend
                                0x400 - 0x7FF -> 0x000 - 0x3FF linksdrehend
*/
unsigned GetDutyCycle(void)
{
    unsigned dc;

    dc = (CCPR1L << 2) + ((CCP1CON & 0x30) >> 4);

    if(direction == LEFT_SPIN)
        dc += 0x400;
    else
        dc = (0x3FF - dc);

    return dc;
}

// Setzt modifizierten DutyCycle: siehe GetDutyCycle()
void SetDutyCycle(unsigned dc)
{
    unsigned duty_cycle;
    if(dc & 0x400 == 0x400)
    {
        duty_cycle = dc & 0x3FF;
        SetDCPWM1(duty_cycle);
        LATCbits.LATC0 = 0;
        LATCbits.LATC1 = 1;
        LATAbits.LATA2 = 0;
        LATAbits.LATA3 = 1;
    }
    else if(dc & 0x400 == 0x000)
    {
        duty_cycle = 0x3FF - dc;
        SetDCPWM1(duty_cycle);
        LATCbits.LATC0 = 1;
        LATCbits.LATC1 = 0;
        LATAbits.LATA2 = 1;
        LATAbits.LATA3 = 0;
    }
}

```

6.7.3 StackTsk.h

```

/*****
 *
 *                      Microchip TCP/IP Stack Definitions for PIC18
 *
 *****/
 *
 * FileName:           StackTsk.h
 * Dependencies:       compiler.h
 * Processor:          PIC18
 * Compiler:           MCC18 v1.00.50 or higher
 *                    HITECH PICC-18 V8.10PL1 or higher
 * Company:            Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") for its PICmicro® Microcontroller is intended and
 * supplied to you, the Company's customer, for use solely and
 * exclusively on Microchip PICmicro Microcontroller products. The
 * software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 * Author              Date      Comment
 * ~~~~~
 * Nilesch Rajbharti   8/10/01 Original      (Rev 1.0)
 * Nilesch Rajbharti   2/9/02 Cleanup
 * Nilesch Rajbharti   5/22/02 Rev 2.0 (See version.log for detail)
 *****/

/*****
 *
 *                      Prozessdatenverarbeitung Vertiefung
 *                      Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 *                      Demoboards und dem Microchip TCP/IP-Stack
 *
 *****/
 *
 *                      Markus Kühle
 *                      Marco Kosinski      534931
 *
 *****/
 *
 *                      Änderungen am Original Quellcode des TCP/IP-Stacks
 *                      StackTsk.h
 *
 *                      TCP, DHCP, FTP, HTTP wurden per define ausgeschaltet
 *                      Das define des MOTOR_SERVERS wurde hinzugefügt
 *****/

#ifndef STACK_TSK_H
#define STACK_TSK_H

#include "compiler.h"

/*
 * This value is used by TCP to implement timeout actions.
 */
#define TICKS_PER_SECOND      (10)

#if (TICKS_PER_SECOND < 10 || TICKS_PER_SECOND > 255)
#error Invalid TICKS_PER_SECONDS specified.
#endif

/*
 * Manually select prescale value to achieve necessary tick period
 * for a given clock frequency.
 */
#define TICK_PRESCALE_VALUE      (256)

#if (TICK_PRESCALE_VALUE != 2 && \
    TICK_PRESCALE_VALUE != 4 && \
    TICK_PRESCALE_VALUE != 8 && \
    TICK_PRESCALE_VALUE != 16 && \
    TICK_PRESCALE_VALUE != 32 && \
    TICK_PRESCALE_VALUE != 64 && \
    TICK_PRESCALE_VALUE != 128 && \
    TICK_PRESCALE_VALUE != 256 )
#error Invalid TICK_PRESCALE_VALUE specified.
#endif

/*
 * This value is for Microchip 24LC256 - 256kb serial EEPROM
 */
#define EEPROM_CONTROL          (0xa0)

/*
 * Number of bytes to be reserved before MPFS storage is to start.
 */

```

```

* These bytes host application configurations such as IP Address,
* MAC Address, and any other required variables.
*
* After making any change to this variable, MPFS.exe must be
* executed with correct block size.
* See MPFS.exe help message by executing MPFS /?
*/
#define MPFS_RESERVE_BLOCK          (32)

/*
* Comment/Uncomment following lines depending on types of modules
* are required.
*/
#define STACK_USE_ICMP
// #define STACK_USE_HTTP_SERVER
#define STACK_USE_MOTOR_SERVER

/*
* For demo purpose only, each sample project defines one or more
* of following defines in compiler command-line options. (See
* each MPLAB Project Node Properties under "Project->Edit Project" menu.
* In real application, user may want to define them here.
*/
// #define STACK_USE_SLIP
// #define STACK_USE_IP_GLEANING
// #define STACK_USE_DHCP
// #define STACK_USE_FTP_SERVER

/*
* Following low level modules are enable/disabled based on high-level
* module selections.
*/
// #define STACK_USE_TCP
#define STACK_USE_UDP

/*
* When SLIP is used, DHCP is not supported.
*/
# if defined(STACK_USE_SLIP)
# undef STACK_USE_DHCP
# endif

/*
* When MPFS_USE_PGRM is used, FTP is not supported.
*/
# if defined(MPFS_USE_PGRM)
# undef STACK_USE_FTP_SERVER
# endif

/*
* Comment following line if StackTsk should wait for acknowledgement
* from remote host before transmitting another packet.
* Commenting following line may reduce throughput.
*/
#define TCP_NO_WAIT_FOR_ACK

/*
* Uncomment following line if this stack will be used in CLIENT
* mode. In CLIENT mode, some functions specific to client operation
* are enabled.
*/
// #define STACK_CLIENT_MODE

/*
* If html pages are stored in internal program memory,
* uncomment MPFS_USE_PGRM and comment MPFS_USE_EEPROM
* If html pages are stored in external eeprom memory,
* comment MPFS_USE_PGRM and uncomment MPFS_USE_EEPROM
*/
// #define MPFS_USE_PGRM
// #define MPFS_USE_EEPROM

# if defined(MPFS_USE_PGRM) && defined(MPFS_USE_EEPROM)
# error Invalid MPFS Storage option specified.
# endif

# if !defined(MPFS_USE_PGRM) && !defined(MPFS_USE_EEPROM)
# error You have not specified MPFS storage option.
# endif

/*
* When HTTP is enabled, TCP must be enabled.
*/
# if defined(STACK_USE_HTTP_SERVER)
# if !defined(STACK_USE_TCP)
# define STACK_USE_TCP
# endif
# endif

/*
* When FTP is enabled, TCP must be enabled.
*/
# if defined(STACK_USE_FTP_SERVER)
# if !defined(STACK_USE_TCP)
# define STACK_USE_TCP
# endif
# endif

# if defined(STACK_USE_FTP_SERVER)
# define STACK_CLIENT_MODE
# endif

```

```

/*
 * When DHCP is enabled, UDP must also be enabled.
 */
#if defined(STACK_USE_DHCP)
    #if !defined(STACK_USE_UDP)
        #define STACK_USE_UDP
    #endif
#endif

/*
 * When IP Gleaning is enabled, ICMP must also be enabled.
 */
#if defined(STACK_USE_IP_GLEANING)
    #if !defined(STACK_USE_ICMP)
        #define STACK_USE_ICMP
    #endif
#endif

/*
 * DHCP requires unfragmented packet size of at least 328 bytes,
 * and while in SLIP mode, our maximum packet size is less than
 * 255. Hence disallow DHCP module while SLIP is in use.
 * If required, one can use DHCP while SLIP is in use by modifying
 * C18 linker script file such that C18 compiler can allocate
 * a static array larger than 255 bytes.
 * Due to very specific application that would require this,
 * sample stack does not provide such facility. Interested users
 * must do this on their own.
 */
#if defined(STACK_USE_SLIP)
    #if defined(STACK_USE_DHCP)
        #error DHCP cannot be used when SLIP is enabled.
    #endif
#endif

/*
 * Modify following macros depending on your interrupt usage
 */
#define ENABLE_INTERRUPTS()          INTCON_GIEH = 1
#define DISABLE_INTERRUPTS()         INTCON_GIEH = 0

/*
 * Default Address information - If not found in data EEPROM.
 */
#define MY_DEFAULT_IP_ADDR_BYTE1      (192)
#define MY_DEFAULT_IP_ADDR_BYTE2      (168)
#define MY_DEFAULT_IP_ADDR_BYTE3      (0)
#define MY_DEFAULT_IP_ADDR_BYTE4      (254)

#define MY_DEFAULT_MASK_BYTE1          (0xff)
#define MY_DEFAULT_MASK_BYTE2          (0xff)
#define MY_DEFAULT_MASK_BYTE3          (0xff)
#define MY_DEFAULT_MASK_BYTE4          (0x00)

#define MY_DEFAULT_GATE_BYTE1          MY_DEFAULT_IP_ADDR_BYTE1
#define MY_DEFAULT_GATE_BYTE2          MY_DEFAULT_IP_ADDR_BYTE2
#define MY_DEFAULT_GATE_BYTE3          MY_DEFAULT_IP_ADDR_BYTE3
#define MY_DEFAULT_GATE_BYTE4          MY_DEFAULT_IP_ADDR_BYTE4

#define MY_DEFAULT_MAC_BYTE1           (0x00)
#define MY_DEFAULT_MAC_BYTE2           (0x04)
#define MY_DEFAULT_MAC_BYTE3           (0xa3)
#define MY_DEFAULT_MAC_BYTE4           (0x00)
#define MY_DEFAULT_MAC_BYTE5           (0x00)
#define MY_DEFAULT_MAC_BYTE6           (0x00)

#define MY_MAC_BYTE1                   AppConfig.MyMACAddr.v[0]
#define MY_MAC_BYTE2                   AppConfig.MyMACAddr.v[1]
#define MY_MAC_BYTE3                   AppConfig.MyMACAddr.v[2]
#define MY_MAC_BYTE4                   AppConfig.MyMACAddr.v[3]
#define MY_MAC_BYTE5                   AppConfig.MyMACAddr.v[4]
#define MY_MAC_BYTE6                   AppConfig.MyMACAddr.v[5]

/*
 * Subnet mask for this node.
 * Must not be all zero's or else this node will never transmit
 * anything !!
 */
#define MY_MASK_BYTE1                   AppConfig.MyMask.v[0]
#define MY_MASK_BYTE2                   AppConfig.MyMask.v[1]
#define MY_MASK_BYTE3                   AppConfig.MyMask.v[2]
#define MY_MASK_BYTE4                   AppConfig.MyMask.v[3]

/*
 * Hardcoded IP address of this node
 * My IP = 10.10.5.10
 *
 * Gateway = 10.10.5.10
 */
#define MY_IP_BYTE1                     AppConfig.MyIPAddr.v[0]
#define MY_IP_BYTE2                     AppConfig.MyIPAddr.v[1]
#define MY_IP_BYTE3                     AppConfig.MyIPAddr.v[2]
#define MY_IP_BYTE4                     AppConfig.MyIPAddr.v[3]

/*
 * Hardcoded Gateway address for this node.
 * This should be changed to match actual network environment.

```

```

*/
#define MY_GATE_BYTE1                AppConfig.MyGateway.v[0]
#define MY_GATE_BYTE2                AppConfig.MyGateway.v[1]
#define MY_GATE_BYTE3                AppConfig.MyGateway.v[2]
#define MY_GATE_BYTE4                AppConfig.MyGateway.v[3]

/*
 * TCP configurations
 * To minimize page update, match number of sockets and
 * HTTP connections with different page sources in a
 * page.
 * For example, if page contains reference to 3 more pages,
 * browser may try to open 4 simultaneous HTTP connections,
 * and to minimize browser delay, set HTTP connections to
 * 4, MAX_SOCKETS to 4 and MAC_TX_BUFFER_COUNT to 4.
 * If you are using ICMP or other applications, you should
 * keep at least one socket available for them.
 */

/*
 * Maximum sockets to be defined.
 * Note that each socket consumes 36 bytes of RAM.
 */
#define MAX_SOCKETS                    (5)

/*
 * Available UDP Socket
 */
#define MAX_UDP_SOCKETS                (2)

#if (MAX_SOCKETS <= 0 || MAX_SOCKETS > 255)
#error Invalid MAX_SOCKETS value specified.
#endif

#if (MAX_UDP_SOCKETS <= 0 || MAX_UDP_SOCKETS > 255 )
#error Invalid MAX_UDP_SOCKETS value specified
#endif

#if !defined(STACK_USE_SLIP)
#define MAC_TX_BUFFER_SIZE            (1024)
#define MAC_TX_BUFFER_COUNT          (1)
#else
/*
 * For SLIP, there can only be one transmit and one receive buffer.
 * Both buffer must fit in one bank. If bigger buffer is required,
 * you must manually locate tx and rx buffer in different bank
 * or modify your linker script file to support arrays bigger than
 * 256 bytes.
 */
#define MAC_TX_BUFFER_SIZE            (250)
#define MAC_TX_BUFFER_COUNT          (1)
#endif
// Rests are Receive Buffers
#define MAC_RX_BUFFER_SIZE            (MAC_TX_BUFFER_SIZE)

#if (MAC_TX_BUFFER_SIZE <= 0 || MAC_TX_BUFFER_SIZE > 1500 )
#error Invalid MAC_TX_BUFFER_SIZE value specified.
#endif

#if ( (MAC_TX_BUFFER_SIZE * MAC_TX_BUFFER_COUNT) > (4* 1024) )
#error Not enough room for Receive buffer.
#endif

/*
 * Maximum numbers of simultaneous HTTP connections allowed.
 * Each connection consumes 10 bytes.
 */
#define MAX_HTTP_CONNECTIONS          (3)

#if (MAX_HTTP_CONNECTIONS <= 0 || MAX_HTTP_CONNECTIONS > 255 )
#error Invalid MAX_HTTP_CONNECTIONS value specified.
#endif

#define AVAILABLE_SOCKETS (MAX_SOCKETS)
#if defined(STACK_USE_HTTP_SERVER)
#define AVAILABLE_SOCKETS2 (AVAILABLE_SOCKETS - MAX_HTTP_CONNECTIONS)
#else
#define AVAILABLE_SOCKETS2 (MAX_SOCKETS)
#endif

#if defined(STACK_USE_FTP_SERVER)
#define AVAILABLE_SOCKETS3 (AVAILABLE_SOCKETS2 - 2)
#else
#define AVAILABLE_SOCKETS3 (AVAILABLE_SOCKETS2)
#endif

#if AVAILABLE_SOCKETS3 < 0
#error Maximum TCP Socket count is not enough.
#error Either increase MAX_SOCKETS or decrease module socket usage.
#endif

#if defined(STACK_USE_DHCP)
#if (MAX_UDP_SOCKETS < 1)
#error Set MAX_UDP_SOCKETS to at least one.
#endif
#endif

typedef enum _BOOL { FALSE = 0, TRUE } BOOL;
typedef unsigned char BYTE;           // 8-bit

```

```

typedef unsigned short int WORD;           // 16-bit
typedef unsigned short long SWORD;        // 24-bit
typedef unsigned long DWORD;              // 32-bit

typedef union _SWORD_VAL
{
    SWORD Val;
    struct
    {
        BYTE LSB;
        BYTE MSB;
        BYTE USB;
    } bytes;
} SWORD_VAL;

typedef union _WORD_VAL
{
    WORD Val;
    BYTE v[2];
} WORD_VAL;

#define LSB(a) ((a).v[0])
#define MSB(a) ((a).v[1])

typedef union _DWORD_VAL
{
    DWORD Val;
    BYTE v[4];
} DWORD_VAL;
#define LOWER_LSB(a) ((a).v[0])
#define LOWER_MSB(a) ((a).v[1])
#define UPPER_LSB(a) ((a).v[2])
#define UPPER_MSB(a) ((a).v[3])

typedef BYTE BUFFER;

typedef struct _MAC_ADDR
{
    BYTE v[6];
} MAC_ADDR;

typedef union _IP_ADDR
{
    BYTE v[4];
    DWORD Val;
} IP_ADDR;

typedef struct _NODE_INFO
{
    MAC_ADDR MACAddr;
    IP_ADDR IPAddr;
} NODE_INFO;

typedef struct _APP_CONFIG
{
    IP_ADDR MyIPAddr;
    MAC_ADDR MyMACAddr;
    IP_ADDR MyMask;
    IP_ADDR MyGateway;
    WORD_VAL SerialNumber;
    IP_ADDR SMTPServerAddr; // Not used.
    struct
    {
        unsigned int bIsDHCPEnabled : 1;
    } Flags;
} APP_CONFIG;

typedef union _STACK_FLAGS
{
    struct
    {
        unsigned int bInConfigMode : 1;
    } bits;
    BYTE Val;
} STACK_FLAGS;

#ifndef THIS_IS_STACK_APPLICATION
extern APP_CONFIG AppConfig;
#endif

#if defined(STACK_USE_IP_GLEANING) || defined(STACK_USE_DHCP)
#ifndef STACK_INCLUDE
extern STACK_FLAGS stackFlags;
#endif
#endif

#if defined(STACK_USE_IP_GLEANING) || defined(STACK_USE_DHCP)
#define StackIsInConfigMode() (stackFlags.bits.bInConfigMode)
#else
#define StackIsInConfigMode() (FALSE)
#endif

/*****
* Function: void StackInit(void)
*
* PreCondition: None
*
* Input: None
*
* Output: Stack and its componentns are initialized
*****/

```

```

*
* Side Effects:    None
*
* Note:           This function must be called before any of the
*                  stack or its component routines be used.
*
*****/
void StackInit(void);

/*****
* Function:        void StackTask(void)
*
* PreCondition:    StackInit() is already called.
*
* Input:           None
*
* Output:          Stack FSM is executed.
*
* Side Effects:    None
*
* Note:           This FSM checks for new incoming packets,
*                  and routes it to appropriate stack components.
*                  It also performs timed operations.
*
*                  This function must be called periodically called
*                  to make sure that timely response.
*
*****/
void StackTask(void);

#endif

```

6.7.4 decoder.h

```

/*****
*
*          Prozessdatenverarbeitung Vertiefung
*          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
*          Demoboards und dem Microchip TCP/IP-Stack
*
*****/
*
*          Markus Kühle
*          Marco Kosinski      534931
*
*****/
*
*          decoder.h
*
*****/

#ifndef _DECODER_H_
#define _DECODER_H_

/*****
*
*          Funktionsdeklarationen
*
*****/

void DecoderInit(void);

#endif

```

6.7.5 decoder.c

```

/*****
*
*          Prozessdatenverarbeitung Vertiefung
*          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
*          Demoboards und dem Microchip TCP/IP-Stack
*
*****/
*
*          Markus Kühle
*          Marco Kosinski      534931
*
*****/
*
*          decoder.c
*
*****/

/*****
*
*          Headerdateien
*
*****/

#include <pl8cxxx.h>
#include "decoder.h"

/*****
*
*          Funktionsdefinitionen
*
*****/

// Initialisiert alle Register, die zur Erfassung von Drehrichtung
// und -geschwindigkeit benötigt werden
void DecoderInit(void)
{
    INTCONbits.INT0IE = 1;    // Enable ExtInt0
    INTCONbits.INT0IF = 0;    // Loeschen ExtInt0 Flags
    INTCON2bits.INTEDG0 = 1;  // Interrupt on Rising Edge
    // Der externe Interrupt 0 ist default High Prio

    INTCON3bits.INT1IE = 1;   // Enable ExtInt1
    INTCON3bits.INT1IP = 1;   // ExtInt1 -> High Prio
    INTCON3bits.INT1IF = 0;   // Loeschen des ExtInt1 Flags
}

```

```

    INTCON2bits.INTEDG0 = 1; // Interrupt on Rising Edge

    // Pin 0 u. 1 des PORTB -> Output
    TRISBbits.TRISB0 = 1;
    TRISBbits.TRISB1 = 1;
}

```

6.7.6 interrupt.h

```

/*****
 *
 *          Prozessdatenverarbeitung Vertiefung
 *          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 *          Demoboards und dem Microchip TCP/IP-Stack
 *
 *****/
 *
 *          Markus Kühle
 *          Marco Kosinski      534931
 *
 *****/
 *
 *          interrupt.c
 *****/

#ifndef _INTERRUPT_H_
#define _INTERRUPT_H_

/*****
 *
 *          Headerdateien
 *****/

#include <pl8cxxx.h>
#include "stacktsk.h"

/*****
 *
 *          Definitionen
 *****/

#define LEFT_SPIN 0
#define RIGHT_SPIN 1

#define DEL_INT_FLAGS() INTCONbits.INT0IF = 0; INTCON3bits.INT1IF = 0

/*****
 *
 *          Externe Globale Variable
 *****/

extern volatile BYTE direction;
extern volatile long speed_ist;

/*****
 *
 *          Deklarationen der Interrupt Service Routinen
 *****/

void DecoderISR0(void);
void DecoderISR1(void);
void Timer0ISR(void);

#endif /* _INTERRUPT_H_ */

```

6.7.7 interrupt.c

```

/*****
 *
 *          Prozessdatenverarbeitung Vertiefung
 *          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 *          Demoboards und dem Microchip TCP/IP-Stack
 *
 *****/
 *
 *          Markus Kühle
 *          Marco Kosinski      534931
 *
 *****/
 *
 *          interrupt.c
 *****/

/*****
 *
 *          Headerdateien
 *****/

#include "interrupt.h"
#include "compiler.h"
#include "motor_server.h"

/*****
 *
 *          Globale Variable
 *****/

// Zeitkonstante zur Berechnung der Drehgeschwindigkeit in U/min
// T = 0.0000002[s](Tosc) * 16[ ](TMR0Prsc) * 65536[ ](16-Bit-Timer)
// 1024[ticks/U] (512 pro Pin an PORTB)
// TMR0Period = (T[s]/60[s/min]) * 1024[ticks/U]
const double TMR0Period = 3.5791394133333344; // [ticks*min/U]

volatile unsigned tick_counter = 0;
volatile unsigned cts_counter = 0;
volatile unsigned dsr_counter = 0;

volatile long speed_ist = 0;
volatile unsigned char direction = LEFT_SPIN;

```

```

/*****
 *      Implementierung der HighPrio Interrupt Vektors
 *****/

// Der HighVector beginnt im Speicher an Adresse 0x0008
#pragma code high_vector = 0x0008
void interrupt_at_high_vector(void)
{
    if(INTCONbits.INT0IF == 1) // Externer Int 0
    {
        _asm GOTO DecoderISR0 _endasm
    }
    else if(INTCON3bits.INT1IF == 1) // Externer Int 1
    {
        _asm GOTO DecoderISR1 _endasm
    }
    else if(INTCONbits.TMR0IF == 1) // Timer 0 Int
    {
        _asm GOTO Timer0ISR _endasm
    }
}
#pragma code

/*****
 *      Definitionen der Interrupt Service Routinen
 *****/

// Routine wird bei Uebergang 0->1 auf Pin0 an PORTB ausgelost
#pragma interrupt DecoderISR0
void DecoderISR0(void)
{
    direction = PORTBbits.RB1; // Wenn Pin1 = 1 -> FORWARD
    tick_counter++;
    if(((++dsr_counter)%64) == 0)
    {
        dsr ^= 1;
        ddsr = 1;
    }
    INTCONbits.INT0IF = 0; // Ruecksetzen des ExtInt0 Flags
}

// Routine wird bei Uebergang 0->1 auf Pin1 an PORTB ausgelost
#pragma interrupt DecoderISR1
void DecoderISR1(void)
{
    direction = !PORTBbits.RB0; // Wenn Pin0 = 1 -> BACKWARD
    tick_counter++;
    if(((++cts_counter)%64) == 0)
    {
        cts ^= 1;
        dcts = 1;
    }
    INTCON3bits.INT1IF = 0; // Ruecksetzen des ExtInt1 Flags
}

// Routine wird beim Ueberlauf des Timers ausgelost
// Berechnet die Drehgeschwindigkeit
#pragma interrupt Timer0ISR
void Timer0ISR(void)
{
    speed_is = (long)((double)tick_counter)/TMR0Period + 0.5); // [ticks]/[ticks*min/U] = U/min
    if(direction == RIGHT_SPIN)
        speed_is = -speed_is;
    tick_counter = 0; // Ruecksetzen des Zaehlers
    INTCONbits.TMR0IF = 0; // Ruecksetzen des Timer0Int Flags
}

```

6.7.8 websrvr.c

```

/*****
 *
 *      Example Web Server Application using Microchip TCP/IP Stack
 *
 *****/
* FileName:      WebSrvr.c
* Dependencies:  string.h
*               usart.h
*               StackTsk.h
*               Tick.h
*               http.h
*               MPFS.h
* Processor:     PIC18
* Compiler:      MCC18 v1.00.50 or higher
*               HITECH PICC-18 V8.10PL1 or higher
* Company:       Microchip Technology, Inc.
*
* Software License Agreement
*
* The software supplied herewith by Microchip Technology Incorporated
* (the "Company") for its PICmicro® Microcontroller is intended and
* supplied to you, the Company's customer, for use solely and
* exclusively on Microchip PICmicro Microcontroller products. The
* software is owned by the Company and/or its supplier, and is
* protected under applicable copyright laws. All rights are reserved.
* Any use in violation of the foregoing restrictions may subject the
* user to criminal sanctions under applicable laws, as well as to
* civil liability for the breach of the terms and conditions of this
* license.
*
* THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR

```

```

* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
*
*
* HiTech PICC18 Compiler Options excluding device selection:
* -FAKELOCAL -G -O -Zg -E -C
*
*
*
* Author          Date          Comment
* ~~~~~
* Nilesh Rajbharti 4/19/01      Original (Rev. 1.0)
* Nilesh Rajbharti 2/09/02      Cleanup
* Nilesh Rajbharti 5/22/02      Rev 2.0 (See version.log for detail)
* Nilesh Rajbharti 7/9/02       Rev 2.1 (See version.log for detail)
* ~~~~~
/*****
*
* Prozessdatenverarbeitung Vertiefung
* Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
* Demoboards und dem Microchip TCP/IP-Stack
*
* ~~~~~
*
* Markus Kühle
* Marco Kosinski 534931
*
* ~~~~~
*
* Änderungen am Original Quellcode des TCP/IP-Stacks
* webservr.h
*
* Es wurden diverse Praeprozessorschalter hinzugefuegt
* Um einige Funktionen, die nicht benoetigt wurden
* auszuklammern. Diese werden mit // -> new gekennzeichnet
*
* Desweiteren wurden die Funktionen MotorInit(), MotorServer()
* MotorShutdown() hinzugefuegt
*
* ~~~~~/

/*
* Following define uniquely defines this file as main
* entry/application in whole project, there should only be one such
* definition and application file must define AppConfig variable as
* described below.
*/
#define THIS_IS_STACK_APPLICATION

#define BAUD_RATE (19200) // bps

#define USART_USE_BRGH_LOW
#define USART_USE_BRGH_LOW
#define SPBRG_VAL ((CLOCK_FREQ/BAUD_RATE)/64) - 1)
#else
#define SPBRG_VAL ((CLOCK_FREQ/BAUD_RATE)/16) - 1)
#endif

#if SPBRG_VAL > 255
#error "Calculated SPBRG value is out of range for current CLOCK_FREQ."
#endif

#include <string.h>

/*
* These headers must be included for required defs.
*/
#include "stacktsk.h"

#include "tick.h"

#if defined(STACK_USE_DHCP)
#include "dhcp.h"
#endif

#if defined(STACK_USE_HTTP_SERVER)
#include "http.h"
#endif

#include "mpfs.h"

#if defined(STACK_USE_FTP_SERVER) && defined(MPFS_USE_EEPROM)
#include "ftp.h"
#endif

#include "xlcd.h"

#if defined(MPFS_USE_EEPROM)
#include "xeprom.h"
#endif

// For debug only.
#if defined(STACK_USE_TCP) // -> new
#include "tcp.h"
#endif

#include "icmp.h"

#include "delay.h"

#if defined(STACK_USE_UDP)

```

```

#include "udp.h"
#endif

#if defined (STACK_USE_MOTOR_SERVER) // -> new
#include "motor_server.h"
#endif

ROM char StartupMsg[] = "MCHPStack v2.11";

#if defined(STACK_USE_DHCP) || defined(STACK_USE_IP_GLEANING)
ROM char DHCPMsg[] = "DHCP/Gleaning...";
#endif

ROM char SetupMsg[] = "Board Setup...";

/*
 * This is used by other stack elements.
 * Main application must define this and initialize it with
 * proper values.
 */
APP_CONFIG AppConfig;

BYTE myDHCPBindCount = 0;
#if defined(STACK_USE_DHCP)
extern BYTE DHCPBindCount;
#else
/*
 * If DHCP is not enabled, force DHCP update.
 */
BYTE DHCPBindCount = 1;
#endif

/*
 * Set configuration fuses for HITECH compiler.
 * For MCC18 compiler, separately linked config.asm file
 * will set the correct fuses.
 */
#if defined(HITECH_C18)
__CONFIG(1, UNPROTECT & HS);
__CONFIG(2, PWRTEN & BORDIS & WDTDIS);
#endif

/*
 * Private helper functions.
 * These may or may not be present in all applications.
 */
static void InitAppConfig(void);

static void InitializeBoard(void);

#if defined(STACK_USE_HTTP_SERVER) // -> new
static void ProcessIO(void);
#endif

void NotifyRemoteUser(void);
static void DisplayIPValue(IP_ADDR *IPVal, BOOL bToLCD);
static void SetConfig(void);

#if defined(MPFS_USE_EEPROM)
static BOOL DownloadMPFS(void);
static void SaveAppConfig(void);
#endif

#if defined(STACK_USE_TCP) // -> new
    #if defined(MCHP_C18)
    #pragma interrupt HighISR save=section(".tmpdata")
    void HighISR(void)
    #elif defined(HITECH_C18)
    #if defined(STACK_USE_SLIP)
    extern void MACISR(void);
    #endif
    void interrupt HighISR(void)
    #endif
{
    TickUpdate();

    #if defined(STACK_USE_SLIP)
    MACISR();
    #endif
}

    #if defined(MCHP_C18)
    #pragma code highVector=0x08
void HighVector (void)
{
    _asm goto HighISR _endasm
}
#pragma code /* return to default code section */
    #endif
#endif

static void USARTPut(BYTE c)
{
    while( !TXSTA_TRMT);
    TXREG = c;
}

static void USARTPutString(BYTE *s)
{

```

```

    BYTE c;

    while( (c = *s++) )
        USARTPut(c);
}

#define USARTIsGetReady()    (PIR1_RCIF)
#define USARTGet()          (RCREG)

/*
 * Main entry point.
 */
void main(void)
{
    TBLPTRU = 0x00;

    /*
     * Initialize any application specific hardware.
     */
    InitializeBoard();

    /*
     * Initialize all stack related components.
     * Following steps must be performed for all applications using
     * PICmicro TCP/IP Stack.
     */

    TickInit();

    /*
     * Following steps must be performed for all applications using
     * PICmicro TCP/IP Stack.
     */
    MPFSInit();

    /*
     * Initialize Stack and application related NV variables.
     */
    InitAppConfig();

    /*
     * Depending on whether internal program memor is used or external
     * EEPROM is used, keep/remove these block.
     */

    /*
     * This implementation, initiates Board setup process if RB5
     * is detected low on startup.
     */
    if ( PORTB_RB5 == 0 )
    {
        XLCDGoto(1, 0);
        XLCDPutROMString(SetupMsg);

        SetConfig();
    }

    StackInit();

#ifdef STACK_USE_HTTP_SERVER
    HTTPInit();
#endif

#ifdef STACK_USE_FTP_SERVER && defined(MPFS_USE_EEPROM)
    FTPInit();
#endif

#ifdef STACK_USE_DHCP || defined(STACK_USE_IP_GLEANING)
    if ( AppConfig.Flags.bIsDHCPEnabled )
    {
        XLCDGoto(1, 0);
        XLCDPutROMString(DHCPMsg);
    }
    else
    {
        /*
         * Force IP address display update.
         */
        myDHCPBindCount = 1;
    }
#ifdef STACK_USE_DHCP
    DHCPDisable();
#endif
#endif

#ifdef STACK_USE_MOTOR_SERVER // -> new
    MotorServerInit();
#endif

    /*
     * Once all items are initialized, go into infinite loop and let
     * stack items execute their tasks.
     * If application needs to perform its own task, it should be
     * done at the end of while loop.
     * Note that this is a "co-operative mult-tasking" mechanism
     * where every task performs its tasks (whether all in one shot
     * or part of it) and returns so that other tasks can do their
     * job.
     * If a task needs very long time to do its job, it must broken

```

```

    * down into smaller pieces so that other tasks can have CPU time.
    */

while(1)
{
    /*
     * This task performs normal stack task including checking
     * for incoming packet, type of packet and calling
     * appropriate stack entity to process it.
     */
    StackTask();

#ifdef STACK_USE_HTTP_SERVER
    /*
     * This is a TCP application. It listens to TCP port 80
     * with one or more sockets and responds to remote requests.
     */
    HTTPServer();
#endif

#ifdef STACK_USE_FTP_SERVER && defined(MPFS_USE_EEPROM)
    FTPServer();
#endif

    /*
     * In future, as new TCP/IP applications are written, it
     * will be added here as new tasks.
     */

    /*
     * Add your application specific tasks here.
     */
#ifdef STACK_USE_MOTOR_SERVER // -> new
    MotorServer();
#endif

#ifdef STACK_USE_HTTP_SERVER // -> new
    ProcessIO();
#endif

    /*
     * For DHCP information, display how many times we have renewed the IP
     * configuration since last reset.
     */
    if ( DHCPBindCount != myDHCPBindCount )
    {
        DisplayIPValue(&AppConfig.MyIPAddr, TRUE);
        myDHCPBindCount = DHCPBindCount;

        if ( AppConfig.Flags.bIsDHCPEnabled )
        {
            XLCDGoto(1, 14);
            if ( myDHCPBindCount < 0x0a )
                XLCDPut(myDHCPBindCount + '0');
            else
                XLCDPut(myDHCPBindCount + 'A');
        }
    }
}

#ifdef STACK_USE_MOTOR_SERVER // -> new
MotorServerShutdown();
#endif
}

// 1234567890123456
ROM char blankLCDLine[] = " ";

static void DisplayIPValue(IP_ADDR *IPVal, BOOL bToLCD)
{
    char IPDigit[8];

    if ( bToLCD )
    {
        /*
         * Erase second line.
         */
        XLCDGoto(1, 0);
        XLCDPutROMString(blankLCDLine);
    }

    /*
     * Rewrite the second line.
     */
    XLCDGoto(1, 0);

    itoa(IPVal->v[0], IPDigit);
    if ( bToLCD )
    {
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
    else
    {
        USARTPutString(IPDigit);
        USARTPut('.');
    }

    itoa(IPVal->v[1], IPDigit);
    if ( bToLCD )
    {
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
}

```

```

else
{
    USARTPutString(IPDigit);
    USARTPut('.');
}

itoa(IPVal->v[2], IPDigit);
if ( bToLCD )
{
    XLCDPutString(IPDigit);
    XLCDPut('.');
}
else
{
    USARTPutString(IPDigit);
    USARTPut('.');
}

itoa(IPVal->v[3], IPDigit);
if ( bToLCD )
    XLCDPutString(IPDigit);
else
    USARTPutString(IPDigit);
}

#endif defined(STACK_USE_HTTP_SERVER) // -> new

static char AN0String[8];
static char AN1String[8];

static void ProcessIO(void)
{
    WORD_VAL ADCResult;

    /*
     * Select AN0 channel, Fosc/32 clock
     */
    ADCON0 = 0b10000001;

    /*
     * Wait for acquisition time.
     * Here, rather than waiting for exact time, a simple wait is
     * used. Real applications requiring high accuracy should
     * calculate exact acquisition time and wait accordingly.
     */
    ADCResult.v[0] = 100;
    while( ADCResult.v[0]-- );

    /*
     * First convert AN0 channel.
     * AN0 is already setup as an analog input.
     */
    ADCON0_GO = 1;

    /*
     * Wait until conversion is done.
     */
    while( ADCON0_GO );

    /*
     * Save the result.
     */
    ADCResult.v[0] = ADRESL;
    ADCResult.v[1] = ADRESH;

    /*
     * Convert 10-bit value into ASCII String.
     */
    itoa(ADCResult.Val, AN0String);

    /*
     * Now, convert AN1 channel.
     *
     * In PICDEM.net board, RA2 thru RA7 should be digital or else
     * LED, LCD and NIC would not operate correctly.
     * Since there is no mode where only AN0 and AN1 be analog inputs
     * while rests are digital pins, we will temporarily switch
     * select a mode where RA2 becomes analog input while we do
     * conversion of RA1. Once conversion is done, we will convert
     * RA2 back to digital pin.
     */
    ADCON1 = 0b10000100;

    /*
     * Select AN1 channel.
     */
    ADCON0 = 0b10001001;

    /*
     * Wait for acquisition time.
     * Here, rather than waiting for exact time, a simple wait is
     * used. Real applications requiring high accuracy should
     * calculate exact acquisition time and wait accordingly.
     */
    ADCResult.v[0] = 100;
    while( ADCResult.v[0]-- );

    /*
     * Start the conversion.
     */
    ADCON0_GO = 1;

    /*

```

```

    * Wait until it is done.
    */
    while( ADCON0_GO );

    /*
    * Save the result.
    */
    ADCResult.v[0] = ADRESL;
    ADCResult.v[1] = ADRESH;

    /*
    * Convert 10-bit value into ASCII String.
    */
    itoa(ADCResult.Val, AN1String);

    /*
    * Reset RA2 pin back to digital output.
    */
    ADCON1      = 0b10001110;      // RA0 as analog input.
}

#endif

/*
 * CGI Command Codes.
 */
#define CGI_CMD_DIGOUT      (0)
#define CGI_CMD_LCDOUT     (1)

/*
 * CGI Variable codes. - There could be 0-255 variables.
 */
#define VAR_LED_D5          (0)
#define VAR_LED_D6          (1)
#define VAR_ANAIN_AN0      (2)
#define VAR_ANAIN_AN1      (3)
#define VAR_DIGIN_RB5      (4)
#define VAR_STROUT_LCD     (5)

/*****
 * Function:      void HTTPExecCmd(BYTE** argv, BYTE argc)
 *
 * PreCondition:  None
 *
 * Input:         argv      - List of arguments
 *                argc      - Argument count.
 *
 * Output:        None
 *
 * Side Effects:  None
 *
 * Overview:      This function is a "callback" from HTTPServer
 *                task. Whenever a remote node performs
 *                interactive task on page that was served,
 *                HTTPServer calls this functions with action
 *                arguments info.
 *                Main application should interpret this argument
 *                and act accordingly.
 *
 *                Following is the format of argv:
 *                If HTTP action was : thank.htm?name=Joe&age=25
 *                argv[0] => thank.htm
 *                argv[1] => name
 *                argv[2] => Joe
 *                argv[3] => age
 *                argv[4] => 25
 *
 *                Use argv[0] as a command identifier and rests
 *                of the items as command arguments.
 *
 * Note:          THIS IS AN EXAMPLE CALLBACK.
 *****/
#if defined(STACK_USE_HTTP_SERVER)

ROM char COMMANDS_OK_PAGE[] = "COMMANDS.CGI";

// Copy string with NULL termination.
#define COMMANDS_OK_PAGE_LEN  (sizeof(COMMANDS_OK_PAGE))

ROM char CMD_UNKNOWN_PAGE[] = "INDEX.HTM";

// Copy string with NULL termination.
#define CMD_UNKNOWN_PAGE_LEN  (sizeof(CMD_UNKNOWN_PAGE))

void HTTPExecCmd(BYTE** argv, BYTE argc)
{
    BYTE command;
    BYTE var;

    /*
    * Design your pages such that they contain command code
    * as a one character numerical value.
    * Being a one character numerical value greatly simplifies
    * the job.
    */
    command = argv[0][0] - '0';

    /*
    * Find out the cgi file name and interpret parameters
    * accordingly
    */
    switch(command)
    {
        case CGI_CMD_DIGOUT:

```

```

/*
 * This DIGOUTS.CGI. Any arguments with this file
 * must be about controlling digital outputs.
 */

/*
 * Identify the parameters.
 * Compare it in upper case format.
 */
var = argv[1][0] - '0';

switch(var)
{
case VAR_LED_D5:
/*
 * This is "D5".
 * Toggle D5.
 */
    LATA3 ^= 1;
    break;

case VAR_LED_D6:
/*
 * This is "D6".
 * Toggle it.
 */
    LATA2 ^= 1;
    break;
}

memcpypgm2ram(argv[0],
    (const ROM char*)COMMANDS_OK_PAGE, COMMANDS_OK_PAGE_LEN);
break;

case CGI_CMD_LCDOUT:
/*
 * Note implemented.
 */
    break;

default:
    memcpypgm2ram((unsigned char*)argv[0],
        (const ROM char*)CMD_UNKNOWN_PAGE, CMD_UNKNOWN_PAGE_LEN);
    break;
}
}
#endif

/*****
 * Function:      WORD HTTPGetVar(BYTE var, WORD ref, BYTE* val)
 *
 * PreCondition:  None
 *
 * Input:         var      - Variable Identifier
 *                ref      - Current callback reference with
 *                        respect to 'var' variable.
 *                val      - Buffer for value storage.
 *
 * Output:        Variable reference as required by application.
 *
 * Side Effects:  None
 *
 * Overview:      This is a callback function from HTTPServer() to
 *                main application.
 *                Whenever a variable substitution is required
 *                on any html pages, HTTPServer calls this function
 *                8-bit variable identifier, variable reference,
 *                which indicates whether this is a first call or
 *                not. Application should return one character
 *                at a time as a variable value.
 *
 * Note:          Since this function only allows one character
 *                to be returned at a time as part of variable
 *                value, HTTPServer() calls this function
 *                multiple times until main application indicates
 *                that there is no more value left for this
 *                variable.
 *                On begining, HTTPGetVar() is called with
 *                ref = HTTP_START_OF_VAR to indicate that
 *                this is a first call. Application should
 *                use this reference to start the variable value
 *                extraction and return updated reference. If
 *                there is no more values left for this variable
 *                application should send HTTP_END_OF_VAR. If
 *                there are any bytes to send, application should
 *                return other than HTTP_START_OF_VAR and
 *                HTTP_END_OF_VAR reference.
 *
 *                THIS IS AN EXAMPLE CALLBACK.
 *                MODIFY THIS AS PER YOUR REQUIREMENTS.
 *****/
#if defined(STACK_USE_HTTP_SERVER)
WORD HTTPGetVar(BYTE var, WORD ref, BYTE* val)
{
    /*
     * First of all identify variable.
     */
    switch(var)
    {
case VAR_LED_D5:
        if ( LATA3 )
            *val = '0';
        else

```

```

        *val = '1';
        break;

case VAR_LED_D6:
    if ( LATA2 )
        *val = '0';
    else
        *val = '1';
    break;

case VAR_ANAIN_AN0:
    if ( ref == HTTP_START_OF_VAR )
    {
        ref = (BYTE)0;
    }
    *val = AN0String[(BYTE)ref];
    if ( AN0String[(BYTE)ref] == '\0' )
        return HTTP_END_OF_VAR;

    (BYTE)ref++;
    return ref;

case VAR_ANAIN_AN1:
    if ( ref == HTTP_START_OF_VAR )
    {
        ref = (BYTE)0;
    }
    *val = AN1String[(BYTE)ref];
    if ( AN1String[(BYTE)ref] == '\0' )
        return HTTP_END_OF_VAR;

    (BYTE)ref++;
    return ref;

case VAR_DIGIN_RB5:
    if ( PORTB_RB5 )
        *val = '1';
    else
        *val = '0';
    break;
}

return HTTP_END_OF_VAR;
}
#endif

#if defined(STACK_USE_FTP_SERVER) && defined(MPFS_USE_EEPROM)
ROM char FTP_USER_NAME[] = "ftp";
#undef FTP_USER_NAME_LEN
#define FTP_USER_NAME_LEN (sizeof(FTP_USER_NAME)-1)

ROM char FTP_USER_PASS[] = "microchip";
#define FTP_USER_PASS_LEN (sizeof(FTP_USER_PASS)-1)

BOOL FTPVerify(char *login, char *password)
{
    if ( !memcmppgm2ram(login, FTP_USER_NAME, FTP_USER_NAME_LEN) )
    {
        if ( !memcmppgm2ram(password, FTP_USER_PASS, FTP_USER_PASS_LEN) )
            return TRUE;
    }
    return FALSE;
}
#endif

/*****
 * Function:      void InitializeBoard(void)
 *
 * PreCondition:  None
 *
 * Input:         None
 *
 * Output:        None
 *
 * Side Effects:  None
 *
 * Overview:      Initialize board specific hardware.
 *
 * Note:          None
 *****/
static void InitializeBoard(void)
{
    /*
     * Setup for PORTA.RA0 as analog input while rests
     * as digital i/o lines.
     */
    ADCON1 = 0b10001110;    // RA0 as analog input, Right justified
    TRISA = 0x03;

    /*
     * LCD is enabled using RA5.
     */
    PORTA_RA5 = 0;          // Disable LCD.

    /*
     * Turn off the LED's.
     */
    // LATA2 = 1;
    // LATA3 = 1;

    /*

```

```

    * External data EEPROM needs pull-ups, so enable internal
    * pull-ups.
    */
    INTCON2_RBPU = 0;

    XLCDInit();
    XLCDGoto(0, 0);
    XLCDPutROMString(StartupMsg);

    TXSTA = 0b00100000;    // Low BRG speed
    RCSTA = 0b10010000;
    SPBRG = SPBRG_VAL;

    TOCON = 0;
    //INTCON_GIEH = 1; // -> new
    //INTCON_GIEL = 1; // -> new
}

/*****
 * Function:      void InitAppConfig(void)
 *
 * PreCondition:  MPFSInit() is already called.
 *
 * Input:         None
 *
 * Output:        Write/Read non-volatile config variables.
 *
 * Side Effects:  None
 *
 * Overview:      None
 *
 * Note:          None
 *****/
static void InitAppConfig(void)
{
    #if defined(MPFS_USE_EEPROM)
    BYTE c;
    BYTE *p;
    #endif

    /*
     * Load default configuration into RAM.
     */
    AppConfig.MyIPAddr.v[0] = MY_DEFAULT_IP_ADDR_BYTE1;
    AppConfig.MyIPAddr.v[1] = MY_DEFAULT_IP_ADDR_BYTE2;
    AppConfig.MyIPAddr.v[2] = MY_DEFAULT_IP_ADDR_BYTE3;
    AppConfig.MyIPAddr.v[3] = MY_DEFAULT_IP_ADDR_BYTE4;

    AppConfig.MyMask.v[0] = MY_DEFAULT_MASK_BYTE1;
    AppConfig.MyMask.v[1] = MY_DEFAULT_MASK_BYTE2;
    AppConfig.MyMask.v[2] = MY_DEFAULT_MASK_BYTE3;
    AppConfig.MyMask.v[3] = MY_DEFAULT_MASK_BYTE4;

    AppConfig.MyGateway.v[0] = MY_DEFAULT_GATE_BYTE1;
    AppConfig.MyGateway.v[1] = MY_DEFAULT_GATE_BYTE2;
    AppConfig.MyGateway.v[2] = MY_DEFAULT_GATE_BYTE3;
    AppConfig.MyGateway.v[3] = MY_DEFAULT_GATE_BYTE4;

    AppConfig.MyMACAddr.v[0] = MY_DEFAULT_MAC_BYTE1;
    AppConfig.MyMACAddr.v[1] = MY_DEFAULT_MAC_BYTE2;
    AppConfig.MyMACAddr.v[2] = MY_DEFAULT_MAC_BYTE3;
    AppConfig.MyMACAddr.v[3] = MY_DEFAULT_MAC_BYTE4;
    AppConfig.MyMACAddr.v[4] = MY_DEFAULT_MAC_BYTE5;
    AppConfig.MyMACAddr.v[5] = MY_DEFAULT_MAC_BYTE6;

    #if defined(STACK_USE_DHCP) || defined(STACK_USE_IP_GLEANING)
    AppConfig.Flags.bIsDHCPEnabled = TRUE;
    #else
    AppConfig.Flags.bIsDHCPEnabled = FALSE;
    #endif

    #if defined(MPFS_USE_EEPROM)
    p = (BYTE*)&AppConfig;

    XEEBeginRead(EEPROM_CONTROL, 0x00);
    c = XEERead();
    XEEEndRead();

    /*
     * When a record is saved, first byte is written as 0x55 to indicate
     * that a valid record was saved.
     */
    if ( c == 0x55 )
    {
        XEEBeginRead(EEPROM_CONTROL, 0x01);
        for ( c = 0; c < sizeof(AppConfig); c++ )
            *p++ = XEERead();
        XEEEndRead();
    }
    else
        SaveAppConfig();
    #endif
    DisplayIPValue(&AppConfig.MyIPAddr, TRUE);
}

#if defined(MPFS_USE_EEPROM)
static void SaveAppConfig(void)
{
    BYTE c;
    BYTE *p;

    p = (BYTE*)&AppConfig;

```

```

XEEBeginWrite(EEPROM_CONTROL, 0x00);
XEEWrite(0x55);
for ( c = 0; c < sizeof(AppConfig); c++ )
{
    XEEWrite(*p++);
}

XEEEndWrite();
}
#endif

ROM char menu[] =
"\r\n\r\n\r\nMCHPStack Demo Application v1.0 (Microchip TCP/IP Stack 2.0)\r\n\r\n"

"\t1: Change Board serial number.\r\n"
"\t2: Change default IP address.\r\n"
"\t3: Change default gateway address.\r\n"
"\t4: Change default subnet mask.\r\n"
"\t5: Enable DHCP & IP Gleaning.\r\n"
"\t6: Disable DHCP & IP Gleaning.\r\n"
"\t7: Download MPFS image.\r\n"
"\t8: Save & Quit.\r\n"
"\r\n"
"Enter a menu choice (1-8): ";

typedef enum _MENU_CMD
{
    MENU_CMD_SERIAL_NUMBER          = '1',
    MENU_CMD_IP_ADDRESS,
    MENU_CMD_GATEWAY_ADDRESS,
    MENU_CMD_SUBNET_MASK,
    MENU_CMD_ENABLE_AUTO_CONFIG,
    MENU_CMD_DISABLE_AUTO_CONFIG,
    MENU_CMD_DOWNLOAD_MPFS,
    MENU_CMD_QUIT,
    MENU_CMD_INVALID
} MENU_CMD;

ROM char* menuCommandPrompt[] =
{
    "\r\nSerial Number (",
    "\r\nDefault IP Address (",
    "\r\nDefault Gateway Address (",
    "\r\nDefault Subnet Mask (",
    "\r\nDHCP & IP Gleaning enabled.\r\n",
    "\r\nDHCP & IP Gleaning disabled.\r\n",
    "\r\nReady to download MPFS image - Use Xmodem protocol.\r\n",
    "\r\nNow running application..."
};

ROM char InvalidInputMsg[] = "\r\nInvalid input received - Input ignored.\r\n"
                             "Press any key to continue...\r\n";

void USARTPutROMString(ROM char* str)
{
    BYTE v;

    while( v = *str++ )
        USARTPut(v);
}

BYTE USARTGetString(char *buffer, BYTE bufferLen)
{
    BYTE v;
    BYTE count;

    count = 0;
    do
    {
        while( !USARTIsGetReady() );

        v = USARTGet();

        if ( v == '\r' || v == '\n' )
            break;

        count++;
        *buffer++ = v;
        *buffer = '\0';
        if ( bufferLen-- == 0 )
            break;
    } while(1);
    return count;
}

BOOL StringToIPAddress(char *str, IP_ADDR *buffer)
{
    BYTE v;
    char *temp;
    BYTE byteIndex;

    temp = str;
    byteIndex = 0;

    while( v = *str )
    {
        if ( v == '.' )
        {
            *str++ = '\0';
            buffer->v[byteIndex++] = atoi(temp);
            temp = str;
        }
        else if ( v < '0' || v > '9' )
            return FALSE;
    }
}

```

```

        str++;
    }

    buffer->v[byteIndex] = atoi(temp);

    return (byteIndex == 3);
}

MENU_CMD GetMenuChoice(void)
{
    BYTE c;

    while ( !USARTIsGetReady() );

    c = USARTGet();

    if ( c >= '1' && c < MENU_CMD_INVALID )
        return c;
    else
        return MENU_CMD_INVALID;
}

#define MAX_USER_RESPONSE_LEN    (20)
void ExecuteMenuChoice(MENU_CMD choice)
{
    char response[MAX_USER_RESPONSE_LEN];
    IP_ADDR tempIPValue;
    IP_ADDR *destIPValue;

    USARTPut('\r');
    USARTPut('\n');
    USARTPutROMString(menuCommandPrompt[choice-'0'-1]);

    switch(choice)
    {
        case MENU_CMD_SERIAL_NUMBER:
            itoa(AppConfig.SerialNumber.Val, response);
            USARTPutString(response);
            USARTPut(' ');
            USARTPut(':');
            USARTPut(' ');

            if ( USARTGetString(response, sizeof(response)) )
            {
                AppConfig.SerialNumber.Val = atoi(response);

                AppConfig.MyMACAddr.v[4] = AppConfig.SerialNumber.v[1];
                AppConfig.MyMACAddr.v[5] = AppConfig.SerialNumber.v[0];
            }
            else
                goto HandleInvalidInput;

            break;

        case MENU_CMD_IP_ADDRESS:
            destIPValue = &AppConfig.MyIPAddr;
            goto ReadIPConfig;

        case MENU_CMD_GATEWAY_ADDRESS:
            destIPValue = &AppConfig.MyGateway;
            goto ReadIPConfig;

        case MENU_CMD_SUBNET_MASK:
            destIPValue = &AppConfig.MyMask;

        ReadIPConfig:
            DisplayIPValue(destIPValue, FALSE);
            USARTPut(' ');
            USARTPut(':');
            USARTPut(' ');

            USARTGetString(response, sizeof(response));

            if ( !StringToIPAddress(response, &tempIPValue) )
            {
                HandleInvalidInput:
                    USARTPutROMString(InvalidInputMsg);
                    while( !USARTIsGetReady() );
                    USARTGet();
            }
            else
            {
                destIPValue->Val = tempIPValue.Val;
            }
            break;

        case MENU_CMD_ENABLE_AUTO_CONFIG:
            AppConfig.Flags.bIsDHCPEnabled = TRUE;
            break;

        case MENU_CMD_DISABLE_AUTO_CONFIG:
            AppConfig.Flags.bIsDHCPEnabled = FALSE;
            break;

        case MENU_CMD_DOWNLOAD_MPFS:
            #if defined(MPFS_USE_EEPROM)
                DownloadMPFS();
            #endif
            break;

        case MENU_CMD_QUIT:

```

```

#if defined(MPFS_USE_EEPROM)
    SaveAppConfig();
#endif
    break;
}
}

static void SetConfig(void)
{
    MENU_CMD choice;

    do
    {
        USARTPutROMString(menu);
        choice = GetMenuChoice();
        if ( choice != MENU_CMD_INVALID )
            ExecuteMenuChoice(choice);
    } while(choice != MENU_CMD_QUIT);
}

#if defined(MPFS_USE_EEPROM)
/*****
 * Function:      BOOL DownloadMPFS(void)
 * PreCondition:  MPFSInit() is already called.
 * Input:         None
 * Output:        TRUE if successful
 *                FALSE otherwise
 * Side Effects:  This function uses 128 bytes of Bank 4 using
 *                indirect pointer. This requires that no part of
 *                code is using this block during or before calling
 *                this function. Once this function is done,
 *                that block of memory is available for general use.
 * Overview:      This function implements XMODEM protocol to
 *                be able to receive a binary file from PC
 *                applications such as HyperTerminal.
 * Note:          In current version, this function does not
 *                implement user interface to set IP address and
 *                other informations. User should create their
 *                own interface to allow user to modify IP
 *                information.
 *                Also, this version implements simple user
 *                action to start file transfer. User may
 *                evaluate its own requirement and implement
 *                appropriate start action.
 *****/
#define XMODEM_SOH      0x01
#define XMODEM_EOT      0x04
#define XMODEM_ACK      0x06
#define XMODEM_NAK      0x15
#define XMODEM_CAN      0x18
#define XMODEM_BLOCK_LEN 128

static BOOL DownloadMPFS(void)
{
    enum SM_MPFS
    {
        SM_MPFS_SOH,
        SM_MPFS_BLOCK,
        SM_MPFS_BLOCK_CMP,
        SM_MPFS_DATA,
    } state;

    BYTE c;
    MPFS handle;
    BOOL lbDone;
    BYTE blockLen;
    BYTE lResult;
    BYTE tempData[XMODEM_BLOCK_LEN];
    TICK lastTick;
    TICK currentTick;

    state = SM_MPFS_SOH;
    lbDone = FALSE;

    handle = MPFSFormat();

    /*
     * Notify the host that we are ready to receive...
     */
    lastTick = TickGet();
    do
    {
        /*
         * Update tick here too - just in case interrupt is not used.
         */
        TickUpdate();

        currentTick = TickGet();
        if ( TickGetDiff(currentTick, lastTick) >= (TICK_SECOND/2) )
        {
            lastTick = TickGet();
            USARTPut(XMODEM_NAK);

```

```

        /*
         * Blink LED to indicate that we are waiting for
         * host to send the file.
         */
        // LATA2 ^= 1;
    }

} while( !USARTIsGetReady() );

while(!lbDone)
{
    /*
     * Update tick here too - just in case interrupt is not used.
     */
    TickUpdate();

    if ( USARTIsGetReady() )
    {
        /*
         * Toggle LED as we receive the data from host.
         */
        // LATA2 ^= 1;
        c = USARTGet();
    }
    else
    {
        /*
         * Real application should put some timeout to make sure
         * that we do not wait forever.
         */
        continue;
    }

    switch(state)
    {
    default:
        if ( c == XMODEM_SOH )
        {
            state = SM_MPFS_BLOCK;
        }
        else if ( c == XMODEM_EOT )
        {
            /*
             * Turn off LED when we are done.
             */
            // LATA2 = 1;

            MPFSClose();
            USARTPut(XMODEM_ACK);
            lbDone = TRUE;
        }
        else
            USARTPut(XMODEM_NAK);

        break;

    case SM_MPFS_BLOCK:
        /*
         * We do not use block information.
         */
        lResult = XMODEM_ACK;
        blockLen = 0;
        state = SM_MPFS_BLOCK_CMP;
        break;

    case SM_MPFS_BLOCK_CMP:
        /*
         * We do not use l's comp. block value.
         */
        state = SM_MPFS_DATA;
        break;

    case SM_MPFS_DATA:
        /*
         * Buffer block data until it is over.
         */
        tempData[blockLen++] = c;
        if ( blockLen > XMODEM_BLOCK_LEN )
        {
            /*
             * We have one block data.
             * Write it to EEPROM.
             */
            MPFSPutBegin(handle);

            lResult = XMODEM_ACK;
            for ( c = 0; c < XMODEM_BLOCK_LEN; c++ )
                MPFSPut(tempData[c]);

            handle = MPFSPutEnd();

            USARTPut(lResult);
            state = SM_MPFS_SOH;
        }
        break;
    }
}

}

/*
 * This small wait is required if SLIP is in use.

```

```

* If this is not used, PC might misinterpret SLIP
* module communication and never close file transfer
* dialog box.
*/
#if defined(STACK_USE_SLIP)
{
    BYTE i;
    i = 255;
    while( i-- );
}
#endif
return TRUE;
}

#endif

void XLCDDelay15ms(void)
{
    DelayMs(15);
}
void XLCDDelay4ms(void)
{
    DelayMs(4);
}

void XLCDDelay100us(void)
{
    //INTCON_GIEH = 0;
    Delay10us(1);
    //INTCON_GIEH = 1;
}

```

6.8 Source Code motor_serial

6.8.1 usart.h

```

/*****
*
*          Prozessdatenverarbeitung Vertiefung
*          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
*          Demoboards und dem Microchip TCP/IP-Stack
*
*****/
*
*          Markus Kühle
*          Marco Kosinski      534931
*
*****/
*          interrupt.c
*****/

#ifndef _USART_H_
#define _USART_H_

/*****
*
*          Definitionen
*****/

#define READ 0
#define WRITE 1

#define BUFF_SIZE 64

#define SPACE 0x20
#define ESC 0x1B
#define CR 0x0D
#define BS 0x08

#define LEFT 'l'
#define RIGHT 'r'
#define DISABLE 'd'
#define BREAK 'b'
#define INT 'i'

/*****
*
*          Funktionsdeklarationen
*****/

unsigned char read_write_buffer(char*, unsigned char);
void USARTInit(void);

#endif /* _USART_H_ */

```

6.8.2 usart.c

```

/*****
*
*          Prozessdatenverarbeitung Vertiefung
*          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
*          Demoboards und dem USART
*
*****/
*
*          Markus Kühle
*          Marco Kosinski      534931
*
*****/
*          interupt.c
*****/

```

```

/*****
 *      Headerdateien
 *****/

#include <p18cxxx.h>
#include "interrupt.h"
#include "usart.h"
#include "defines.h"

/*****
 *      Funktionsdefinitionen
 *****/

// Die Funktion realisiert einen ringförmigen Puffer, der
// via Flag schreibend oder lesend aufgerufen werden kann
unsigned char read_write_buffer(char *c, unsigned char rw)
{
    volatile static char buffer[BUFF_SIZE]; // Statischer Puffer
    volatile static unsigned read_ptr = 0;
    volatile static unsigned write_ptr = 0;

    MASK_USART_RX_INT;

    if(rw == READ)
    {
        if(read_ptr != write_ptr) // Puffer enthält Zeichen
        {
            *c = buffer[read_ptr++];
            if(read_ptr >= BUFF_SIZE)
                read_ptr = 0;
            UMASK_USART_RX_INT;
            return TRUE;
        }
        else // Puffer leer
        {
            UMASK_USART_RX_INT;
            return FALSE;
        }
    }
    else // rw == WRITE
    {
        buffer[write_ptr++] = *c;
        if(write_ptr >= BUFF_SIZE)
            write_ptr = 0;
        UMASK_USART_RX_INT;
        return TRUE;
    }
}

// Initialisiert alle benoetigten USART-Register
void USARTInit(void)
{
    IPRbits.RCIP    = 1; // Make USART RX Interrupt High Prio

    SPBRG = 15;
    TXSTAbits.BRGH  = 0; // Baudrate Low Speed select
    TXSTAbits.SYNC  = 0; // Asynchroner Modus
    RCSTAbits.SPEN  = 1; // Serial Port Enabled
    PIRbits.RCIE    = 1; // Enable USART RX Interrupt
    RCSTAbits.RX9   = 0; // 8-Bit Empfang
    RCSTAbits.CREN  = 1; // Enable Receiver
}

```

6.8.3 xlcd.h

```

/*****
 *
 *      External LCD access routines defs
 *
 *****/
* FileName:      XLCD.h
* Dependencies:  compiler.h
* Processor:     PIC18
* Compiler:      MCC18 v1.00.50 or higher
*                HITECH PICC-18 V8.10PL1 or higher
* Company:       Microchip Technology, Inc.
*
* Software License Agreement
*
* The software supplied herewith by Microchip Technology Incorporated
* (the "Company") for its PICmicro® Microcontroller is intended and
* supplied to you, the Company's customer, for use solely and
* exclusively on Microchip PICmicro Microcontroller products. The
* software is owned by the Company and/or its supplier, and is
* protected under applicable copyright laws. All rights are reserved.
* Any use in violation of the foregoing restrictions may subject the
* user to criminal sanctions under applicable laws, as well as to
* civil liability for the breach of the terms and conditions of this
* license.
*
* THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
*
* Author      Date      Comment
* ~~~~~~
* Nilesh Rajbharti  5/8/02  Original          (Rev 1.0)
* Nilesh Rajbharti  7/10/02 Improved
 *****/
#ifndef __XLCD_H
#define __XLCD_H

```

```

/* XLCD peripheral routines.
 *
 * Notes:
 * - These libraries routines are written to support the
 *   Hitachi HD44780 LCD controller.
 * - The user must define the following items:
 *   - The LCD interface type (4- or 8-bits)
 *   - If 4-bit mode
 *     - whether using the upper or lower nibble
 *   - The data port
 *     - The tris register for data port
 *     - The control signal ports and pins
 *     - The control signal port tris and pins
 * - The user must provide three delay routines:
 *   - DelayFor18TCY() provides a 18 Tcy delay
 *   - DelayPORXLCD() provides at least 15ms delay
 *   - DelayXLCD() provides at least 5ms delay
 */

#define DATA_PORT PORTD
#define TRIS_DATA_PORT TRISD

/*
 * Set your LCD type.
 */
#define XCLD_TYPE (FOUR_BIT & LINES_5X8)

/*
 * This is how LCD will be setup on Init.
 */
#define XLCD_DISPLAY_SETUP (CURSOR_OFF & BLINK_OFF)

/*
 * Uncomment following line if LCD data is to be read back.
 */
// #define XLCD_ENABLE_LCD_READS

/* CTRL_PORT defines the port where the control lines are connected.
 * These are just samples, change to match your application.
 */

#define RW_PIN LATDbits.LATD5 /* PORT for RW */
#define TRIS_RW TRISDbits.TRISD5 /* TRIS for RW */
#define RS_PIN LATDbits.LATD4 /* PORT for RS */
#define TRIS_RS TRISDbits.TRISD4 /* TRIS for RS */
#define E_PIN LATAbits.LATA5 /* PORT for E */
#define TRIS_E TRISAbits.TRISA5 /* TRIS for E */

/* Display ON/OFF Control defines */
#define DON 0b00001111 /* Display on */
#define DOFF 0b00001011 /* Display off */
#define CURSOR_ON 0b00001111 /* Cursor on */
#define CURSOR_OFF 0b00001101 /* Cursor off */
#define BLINK_ON 0b00001111 /* Cursor Blink */
#define BLINK_OFF 0b00001110 /* Cursor No Blink */

/* Cursor or Display Shift defines */
#define SHIFT_CUR_LEFT 0b00010011 /* Cursor shifts to the left */
#define SHIFT_CUR_RIGHT 0b00010111 /* Cursor shifts to the right */
#define SHIFT_DISP_LEFT 0b00011011 /* Display shifts to the left */
#define SHIFT_DISP_RIGHT 0b00011111 /* Display shifts to the right */
#define DISP_CLEAR 0b00000001

/* Function Set defines */
#define FOUR_BIT 0b00101111 /* 4-bit Interface */
#define EIGHT_BIT 0b00111111 /* 8-bit Interface */
#define LINE_5X7 0b00110011 /* 5x7 characters, single line */
#define LINE_5X10 0b00110111 /* 5x10 characters */
#define LINES_5X7 0b00111111 /* 5x7 characters, multiple line */
#define LINES_5X8 0b00111000 /* 5x8 characters, multiple line */

#define ROM rom

extern unsigned CursorAddr;

void XLCDInit(void);
void XLCDPut(char data);
void XLCDPutString(char *string);
void XLCDPutROMString(ROM char *string);
char XLCDIsBusy(void);
void XLCDCommand(unsigned char cmd);
unsigned char XLCDGetAddr(void);
char XLCDGet(void);
void XLCDGetCursorPos(unsigned char *row, unsigned char *col);
void XLCDGoto(unsigned char row, unsigned char col);
// #define XLCDGoto(row, col) XLCDCommand((col + (row * 0x40)) | 0x80)
#define XLCDClear() XLCDCommand(DISP_CLEAR)
#define XLCDCursorRight() XLCDCommand(SHIFT_CUR_RIGHT); CursorAddr++
#define XLCDCursorLeft() XLCDCommand(SHIFT_CUR_LEFT); CursorAddr--
#define XLCDShiftLeft() XLCDCommand(SHIFT_DISP_LEFT); CursorAddr--
#define XLCDShiftRight() XLCDCommand(SHIFT_DISP_RIGHT); CursorAddr++

#define XLCDGRAMAddr(addr) XLCDCommand(addr | 0b01000000)
#define XLCDDRAMAddr(addr) XLCDCommand(addr | 0b10000000)

/* User defines these routines/macros according to the oscillator frequency */
#define XLCDDelay500ns() XLCDDelay100us()

void XLCDDelay15ms(void);
void XLCDDelay4ms(void);
void XLCDDelay100us(void);

#endif

```

6.8.4 xlcd.c

```

/*****
 *
 * External LCD access routines
 *
 *****/
 *
 * FileName:      XLCD.c
 * Dependencies:  xlcd.h
 * Processor:     PIC18
 * Compiler:      MCC18 v1.00.50 or higher
 *               HITECH PICC-18 V8.10PL1 or higher
 * Company:       Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") for its PICmicro® Microcontroller is intended and
 * supplied to you, the Company's customer, for use solely and
 * exclusively on Microchip PICmicro Microcontroller products. The
 * software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 * HiTech PICC18 Compiler Options excluding device selection:
 *      -FAKELOCAL -G -E -C
 *
 *
 *
 *
 * Author          Date      Comment
 * ~~~~~
 * Nilesch Rajbharti  5/8/02  Original      (Rev 1.0)
 * Nilesch Rajbharti  7/10/02 Optimized
 *****/

/*****
 *
 * Prozessdatenverarbeitung Vertiefung
 * Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 * Demoboards und dem USART
 *
 *****/
 *
 * Markus Kühle
 * Marco Kosinski      534931
 *
 *****/
 *
 * Änderungen am Original Quellcode des LCD-Displays
 *
 * unsigned int CursorAddr      - Globale Variable, die die aktuelle
 *                               Adresse im LCD speichert
 *
 * XLCDPut()                    - Wurde um Zeilenumbruchfunktion
 *                               erweitert
 *
 * XLCDGetCursorPos()           - Neue Funktion, die die aktuelle
 *                               Position des Cursors im LCD in
 *                               zwei übergebenen Variablen speichert
 *
 * XLCDGoto()                   - ehemals ein Makro, ändert nun auch
 *                               den Wert von CursorAddr
 *
 *****/

#include "xlcd.h"
#include <p18f452.h>

/*****
 * Function:      void XLCDInit(void)
 *
 * PreCondition:  None
 *
 * Input:         None
 *
 * Output:        None
 *
 * Side Effects:  None
 *
 * Overview:      LCD is intialized
 *
 * Note:          This function will work with all Hitachi HD447780
 *                LCD controller.
 *****/

unsigned int CursorAddr = 0;

void XLCDInit(void)
{
    // The data bits must be either a 8-bit port or the upper or
    // lower 4-bits of a port. These pins are made into inputs

    DATA_PORT      &= 0xf0;
    TRIS_DATA_PORT  |= 0x0f;

    TRIS_RW         = 0;    // All control signals made outputs

```

```

    TRIS_RS      = 0;
    TRIS_E       = 0;
    RW_PIN       = 0;    // R/W pin made low
    RS_PIN       = 0;    // Register select pin made low
    E_PIN        = 0;    // Clock pin made low

    // Delay for 15ms to allow for LCD Power on reset
    XLCDDelay15ms();

    // Setup interface to LCD
    TRIS_DATA_PORT &= 0xf0;
    DATA_PORT    &= 0xf0;
    DATA_PORT    |= 0b00000010;    // Function set cmd(4-bit interface)

    E_PIN = 1;                // Clock the cmd in
    XLCDDelay500ns();
    E_PIN = 0;

    // Delay for at least 4.1ms
    XLCDDelay4ms();

    // Setup interface to LCD
    DATA_PORT &= 0xf0;    // Function set cmd(4-bit interface)
    DATA_PORT |= 0b00000010;

    E_PIN = 1;                // Clock the cmd in
    XLCDDelay500ns();
    E_PIN = 0;

    // Delay for at least 100us
    XLCDDelay100us();

    DATA_PORT    &= 0xf0;    // Function set cmd(4-bit interface)
    DATA_PORT    |= 0b00000010;

    E_PIN        = 1;        // Clock cmd in
    XLCDDelay500ns();
    E_PIN        = 0;

    TRIS_DATA_PORT |= 0x0f;    // Make data nibble input

    // Set data interface width, # lines, font
    XLCDCommand(XLCD_TYPE);    // Function set cmd

    // Set DD Ram address to 0
    XLCDCommand(XLCD_TYPE);

    XLCDCommand(DOFF&XLCD_DISPLAY_SETUP);

    XLCDCommand(DON&XLCD_DISPLAY_SETUP);

    // Clear display
    XLCDCommand(0x01);    // Clear display

    // Set entry mode inc, no shift
    XLCDCommand(SHIFT_CUR_LEFT);    // Entry Mode

    // Set DD Ram address to 0
    XLCDCommand(0x80);

    return;
}

/*****
 * Function:      void XLCDCommand(unsigned char cmd)
 *
 * PreCondition:  XLCDIsBusy() == FALSE if !defined(XLCD_IS_BLOCKING)
 *
 * Input:         cmd - Command to be set to LCD.
 *
 * Output:        None
 *
 * Side Effects:  None
 *
 * Overview:      None
 *
 * Note:          None
 *****/
void XLCDCommand(unsigned char cmd)
{
    TRIS_RW      = 0;    // All control signals made outputs
    TRIS_RS      = 0;

    TRIS_DATA_PORT &= 0xf0;
    DATA_PORT    &= 0xf0;
    DATA_PORT    |= (cmd>>4);

    RW_PIN = 0;    // Set control signals for command
    RS_PIN = 0;
    XLCDDelay500ns();
    E_PIN = 1;    // Clock command in
    XLCDDelay500ns();
    E_PIN = 0;

    DATA_PORT &= 0xf0;
    DATA_PORT |= cmd&0x0f;

    XLCDDelay500ns();
    E_PIN = 1;    // Clock command in
    XLCDDelay500ns();
    E_PIN = 0;
}

```

```

    TRIS_DATA_PORT |= 0x0f;

    return;
}

/*****
 * Function:      char XLCDIsBusy(void)
 *
 * PreCondition:  None
 *
 * Input:         None
 *
 * Output:        non-zero if LCD controller is ready to accept new
 *                data or command
 *                zero otherwise.
 *
 * Side Effects:   None
 *
 * Overview:      None
 *
 * Note:          None
 *****/
char XLCDIsBusy(void)
{
    RW_PIN = 1;          // Set the control bits for read
    RS_PIN = 0;

    TRIS_RW      = 0;    // All control signals made outputs
    TRIS_RS      = 0;

    XLCDDelay500ns();
    E_PIN = 1;          // Clock in the command
    XLCDDelay500ns();

    if(DATA_PORT&0x08)
    {
        E_PIN = 0;      // Reset clock line
        XLCDDelay500ns();
        E_PIN = 1;      // Clock out other nibble
        XLCDDelay500ns();
        E_PIN = 0;
        RW_PIN = 0;      // Reset control line
        return 1;        // Return TRUE
    }
    else                // Busy bit is low
    {
        E_PIN = 0;      // Reset clock line
        XLCDDelay500ns();
        E_PIN = 1;      // Clock out other nibble
        XLCDDelay500ns();
        E_PIN = 0;
        RW_PIN = 0;      // Reset control line
        return 0;        // Return FALSE
    }
}

/*****
 * Function:      char XLCDGet(void)
 *
 * PreCondition:   XLCDIsBusy() == FALSE && !defined(XLCD_IS_BLOCKING)
 *
 * Input:         None
 *
 * Output:        Current data byte from LCD
 *
 * Side Effects:   None
 *
 * Overview:      None
 *
 * Note:          The data is read from the character generator
 *                RAM or display RAM depending on current setup.
 *****/
char XLCDGet(void)
{
    char data;

    TRIS_RW      = 0;    // All control signals made outputs
    TRIS_RS      = 0;

    RW_PIN = 1;
    RS_PIN = 1;
    XLCDDelay500ns();
    E_PIN = 1;          // Clock the data out of the LCD
    XLCDDelay500ns();

    data = (DATA_PORT<<4)&0xf0; // read the upper nibble of data

    E_PIN = 0;          // Reset the clock line
    XLCDDelay500ns();
    E_PIN = 1;          // Clock the next nibble out of the LCD
    XLCDDelay500ns();

    data |= DATA_PORT&0x0f;    // Read the lower nibble of data

    E_PIN = 0;
    RS_PIN = 0;          // Reset the control bits
    RW_PIN = 0;

    return(data);          // Return the data byte
}

void XLCDPutString(char *string)
{

```

```

    char v;

    while( v = *string )
    {
        XLCDPut(v);
        string++;
    }
}

void XLCDPutROMString(ROM char *string)
{
    char v;

    while( v = *string )
    {
        XLCDPut(v);
        string++;
    }
}

void XLCDDelay4ms(void)
{
    int i;
    for(i = 0; i < 40; i++)
        XLCDDelay100us();
}

void XLCDDelay100us(void)
{
    int i;
    for(i = 0; i < 200; i++)
        Nop();
}

void XLCDDelay15ms(void)
{
    int i;
    for(i = 0; i < 150; i++)
        XLCDDelay100us();
}

/*****
* Function:      void XLCDPut(char data)
* PreCondition:  XLCDInit() is already called AND
*                (XLCDIsBusy() == FALSE AND !defined(XLCD_IS_BLOCKING))
*
* Input:         data    - Data to be written
*
* Output:        None
*
* Side Effects:  None
*
* Overview:      None
*
* Note:          Data is written to character generator RAM or
*                display data RAM depending on how the access is
*                setup.
*****/
void XLCDPut(char data)
{
    if(CursorAddr == 0x10) // LCD hat 16 Stellen pro Reihe
        XLCDGoto(1, 0);
    if(CursorAddr == 0x20)
    {
        XLCDGoto(0, 0);
        CursorAddr = 0x00;
    }
    CursorAddr++;

    TRIS_RW      = 0;    // All control signals made outputs
    TRIS_RS      = 0;

    TRIS_DATA_PORT &= 0xf0;
    DATA_PORT &= 0xf0;
    DATA_PORT |= ((data>>4)&0x0f);

    RS_PIN = 1;          // Set control bits
    RW_PIN = 0;
    XLCDDelay500ns();
    E_PIN = 1;           // Clock nibble into LCD
    XLCDDelay500ns();
    E_PIN = 0;

    DATA_PORT &= 0xf0;
    DATA_PORT |= (data&0x0f);

    XLCDDelay500ns();
    E_PIN = 1;           // Clock nibble into LCD
    XLCDDelay500ns();
    E_PIN = 0;

    TRIS_DATA_PORT |= 0x0f;

    return;
}

/*****
* Function:      unsigned char XLCDGetAddr(void)
*
* PreCondition:  XLCDIsBusy() == FALSE && !defined(XLCD_IS_BLOCKING)
*
* Input:         None
*****/

```

```

*
* Output:          Current address byte from LCD
*
* Side Effects:    None
*
* Overview:        None
*
* Note:            The address is read from the character generator
*                  RAM or display RAM depending on current setup.
*****
unsigned char XLCDGetAddr(void)
{
    char data;      // Holds the data retrieved from the LCD

    TRIS_RW         = 0;    // All control signals made outputs
    TRIS_RS         = 0;

    data = (DATA_PORT<<4)&0xf0; // Read the nibble into the upper nibble of data

    E_PIN = 0;          // Reset the clock
    XLCDDelay500ns();
    E_PIN = 1;          // Clock out the lower nibble
    XLCDDelay500ns();

    data |= DATA_PORT&0x0f;    // Read the nibble into the lower nibble of data

    E_PIN = 0;
    RW_PIN = 0;              // Reset the control lines

    return (data&0x7f);       // Return the address, Mask off the busy bit
}

void XLCDGetCursorPos(unsigned char *row, unsigned char *col)
{
    *row = CursorAddr >> 4;
    *col = CursorAddr &0x0F;
}

void XLCDGoto(unsigned char row, unsigned char col)
{
    CursorAddr = col + (row << 4);
    XLCDCommand((col + (row * 0x40)) | 0x80);
}

```

6.8.5 defines.h

```

/*****
*
*          Prozessdatenverarbeitung Vertiefung
*          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
*          Demoboards und dem USART
*
*****
*
*          Markus Kühle
*          Marco Kosinski 534931
*
*****
*          defines.h
*****/

#ifndef DEFINES_H
#define DEFINES_H

/*****
*
*          Headerdateien
*****/

#include <p18cxxx.h>    // p18cxxx.h must have current processor defined.
#include <stdlib.h>

/*****
*
*          Definitionen
*****/

// Define für C18 Compiler
#define MCHP_C18 // für externe Module aus Stack noch defined gelassen
#define TRUE 1
#define FALSE 0

#define CHAR_MAX 16

/*
* Clock frequency value.
* This value is used to calculate Tick Counter value
*/
#define CLOCK_FREQ          (20000000)    // Hz

#define ROM                  rom

```

```

/*****
 *          Typdefinitionen
 *****/

// Datentypen festlegen
typedef unsigned char BYTE;           // 8-bit
typedef unsigned short int WORD;      // 16-bit

// um ein 16-Bit Wort mit 2 8-Bit Wörtern schreiben zu können
typedef union _WORD_VAL
{
    WORD Val;
    BYTE v[2];
} WORD_VAL;

#endif

```

6.8.6 decoder.h

```

/*****
 *
 *          Prozessdatenverarbeitung Vertiefung
 *          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 *          Demoboards und dem USART
 *
 *****/

Markus Kühle
Marco Kosinski 534931

decoder.h

/*****
 *
 *          Funktionsdeklarationen
 *****/

void DecoderInit(void);

#endif

```

6.8.7 decoder.c

```

/*****
 *
 *          Prozessdatenverarbeitung Vertiefung
 *          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 *          Demoboards und dem USART
 *
 *****/

Markus Kühle
Marco Kosinski 534931

decoder.c

/*****
 *
 *          Headerdateien
 *****/

#include <pl8cxxx.h>
#include "decoder.h"

/*****
 *
 *          Funktionsdefinitionen
 *****/

// Initialisiert alle Register, die zur Erfassung von Drehrichtung
// und -geschwindigkeit benötigt werden
void DecoderInit(void)
{
    INTCONbits.INT0IE = 1;    // Enable ExtInt0
    INTCONbits.INT0IF = 0;    // Loeschen ExtInt0 Flags
    INTCON2bits.INTEDG0 = 1;  // Interrupt on Rising Edge
    // Der externe Interrupt 0 ist default High Prio

    // Pin 0 u. 1 des PORTB -> Output
    TRISBbits.TRISB0 = 1;
    TRISBbits.TRISB1 = 1;
}

```

}

6.8.8 interrupt.h

```

/*****
 *
 *          Prozessdatenverarbeitung Vertiefung
 *          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 *          Demoboards und dem Microchip TCP/IP-Stack
 *
 *****/
 *
 *          Markus Kühle
 *          Marco Kosinski      534931
 *
 *****/
 *
 *          interrupt.c
 *****/

#ifndef _INTERRUPT_H_
#define _INTERRUPT_H_

/*****
 *
 *          Headerdateien
 *****/

#include <p18cxxx.h>

/*****
 *
 *          Definitionen
 *****/

#define LEFT_SPIN 0
#define RIGHT_SPIN 1

/*****
 *
 *          Maskierungsmakros
 *****/

#define MASK_USART_RX_INT  INTCONbits.GIEH = 0;if(usart_int_count++ == 0)PIELbits.RCIE = 0;INTCONbits.GIEH = 1
#define UMASK_USART_RX_INT INTCONbits.GIEH = 0;if(--usart_int_count == 0)PIELbits.RCIE = 1;INTCONbits.GIEH = 1

#define MASK_DECODER_INT  INTCONbits.GIEH = 0;if(dec_int_count++ == 0)INTCONbits.INT0IE = 0;INTCONbits.GIEH = 1
#define UMASK_DECODER_INT INTCONbits.GIEH = 0;if(--dec_int_count == 0)INTCONbits.INT0IE = 1;INTCONbits.GIEH = 1

#define MASK_TIMER0_INT  INTCONbits.GIEH = 0;if(tmr_int_count++ == 0)INTCONbits.TMR0IE = 0;INTCONbits.GIEH = 1
#define UMASK_TIMER0_INT INTCONbits.GIEH = 0;if(--tmr_int_count == 0)INTCONbits.TMR0IE = 1;INTCONbits.GIEH = 1

/*****
 *
 *          Externe Globale Variable
 *****/

extern volatile unsigned usart_int_count;
extern volatile unsigned dec_int_count;
extern volatile unsigned tmr_int_count;

extern volatile unsigned char direction;
extern volatile long speed;

/*****
 *
 *          Funktionsdeklarationen
 *****/

void USARTrxISR(void);
void DecoderISR(void);
void Timer0ISR(void);

#endif /* _INTERRUPT_H_ */

```

6.8.9 interrupt.c

```

/*****
 *
 *          Prozessdatenverarbeitung Vertiefung
 *          Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 *          Demoboards und dem USART
 *
 *****/
 *
 *          Markus Kühle      137544
 *          Marco Kosinski    534931
 *
 *****/
 *
 *          interupt.c
 *****/

/*****
 *
 *          Headerdateien
 *****/

#include "interrupt.h"
#include "defines.h"
#include "usart.h"

/*****
 *
 *          Globale Variable
 *****/

// Zeitkonstante zur Berechnung der Drehgeschwindigkeit in U/min
// T = 0.0000002[s](Tosc) * 16[ ](TMR0Prsc) * 65536[ ](16-Bit-Timer)
// 512 [ticks/U]
// TMR0Period = (T[s]/60[s/min]) * 1024[ticks/U]

```

```

const double TMR0Period = 1.7895697066666672; // [ticks*min/U]

// Diese Zaehler werden fuer die Maskierungsmacros gebraucht
volatile unsigned usart_int_count = 0;
volatile unsigned dec_int_count = 0;
volatile unsigned tmr_int_count = 0;

// Lochscheibenzaehler
volatile unsigned tick_counter = 0;

volatile long speed = 0;
volatile unsigned char direction = LEFT_SPIN;

/*****
 *
 *      Implementierung des Interrupt Vektors
 *
 *****/

#pragma code high_vector = 0x08
void interrupt_at_high_vector(void)
{
    if(INTCONbits.INT0IF == 1) // Externer Int 0
    {
        _asm GOTO DecoderISR _endasm
    }
    else if(INTCONbits.TMR0IF == 1) // Timer 0 Int
    {
        _asm GOTO Timer0ISR _endasm
    }
    else if(PIR1bits.RCIF == 1) // Recv Int
    {
        _asm GOTO USARTrxISR _endasm
    }
}
#pragma code

/*****
 *
 *      Interrupt Service Routinen
 *
 *****/

// Der Interrupt wird ausgeloezt, wenn ein Zeichen empfangen wurde
#pragma interrupt USARTrxISR
void USARTrxISR(void)
{
    char data;

    while(PIR1bits.RCIF)
    {
        data = RCREG;
        read_write_buffer(&data, WRITE);
    }
}

// Routine wird bei Uebergang 0->1 auf Pin0 an PORTB ausgeloezt
#pragma interrupt DecoderISR
void DecoderISR(void)
{
    direction = PORTBbits.RB1;
    tick_counter++;
    INTCONbits.INT0IF = 0;
}

// Routine wird beim Ueberlauf des Timers ausgeloezt
// Berechnet die Drehgeschwindigkeit
#pragma interrupt Timer0ISR
void Timer0ISR(void)
{
    speed = (long)((double)tick_counter/TMR0Period); // [ticks]/[ticks*min/U] = U/min
    if(direction == RIGHT_SPIN)
        speed = -speed;
    tick_counter = 0;
    INTCONbits.TMR0IF = 0;
}

```

6.8.10 motor_serial.c

```

/*****
 *
 *      Prozessdatenverarbeitung Vertiefung
 *      Ansteuerung eines Motors über UDP mit Hilfe des PICDEM.net
 *      Demoboards und dem USART
 *
 *****/

*
*      Markus Kühle
*      Marco Kosinski      534931
*
*****/

*
*      motor_serial.c
*
*****/

/*****
 *
 *      Headerdateien
 *
 *****/

#include <p18cxxx.h>
#include <stdlib.h>
#include <ctype.h>
#include <timers.h>
#include <string.h>
#include <pwm.h>
#include "interrupt.h"
#include "decoder.h"
#include "usart.h"

```

```

#include "xlcd.h"
#include "defines.h"

/*****
 *           Globale Variable
 *****/

//           1234567890123456
ROM char blankLCDLine[] = "          ";

/*****
 *           Main-Funktion
 *****/

int main(void)
{
    char data;
    char xlcd_buffer_higher[CHAR_MAX]; // Obere LCD Zeile
    char xlcd_buffer_lower[CHAR_MAX]; // Untere LCD Zeile
    unsigned char_pointer = 0;
    unsigned duty_cycle;
    unsigned char row;
    unsigned char col;
    char mode = LEFT;

    RCONbits.IPEN = 1; // Enable Priority Interrupts
    INTCONbits.GIEH = 0; // Disable High Prio Interrupts
    INTCONbits.GIEL = 0; // Disable Low Prio Interrupts

    INTCON2bits.RBPU = 1; // Disable Pullups auf PORTB

    // Initilisieren der Peripherie
    USARTInit();
    XLCDInit();
    DecoderInit();

    OpenTimer0(TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_16);
    OpenTimer2(TIMER_INT_OFF & T2_PS_1_16 & T2_POST_1_1); // Prescaler fuer PWM
    OpenPWM1(0xFF);
    SetDCPWM1(0x00);

    // Interrupts einschalten
    INTCONbits.GIEH = 1;
    INTCONbits.GIEL = 1;

    // Benoetigte PortPins als Ausgang schalten
    TRISCbits.TRISC0 = 0; // Drehrichtung Rechts
    TRISCbits.TRISC1 = 0; // Drehrichtung Links
    TRISCbits.TRISC2 = 0; // Ausgang PWM
    TRISAbits.TRISA2 = 0; // LED Links
    TRISAbits.TRISA3 = 0; // LED Rechts

    // Drehricht auf Links Initialisieren
    LATCbits.LATC0 = 0;
    LATCbits.LATC1 = 1;
    LATAbits.LATA2 = 0; // Leuchtet bei 0
    LATAbits.LATA3 = 1; // Leuchtet bei 0

    XLCDClear();

    while(1)
    {
        if(read_write_buffer(&data, READ)) // Falls Zeichen im Puffer bereitsteht
        {
            if(isalpha(data))
            {
                switch(data)
                {
                    case BREAK: // In BREAK-Mode wechseln
                        if(mode == DISABLE)
                            XLCDClear();
                        mode = BREAK;
                        LATCbits.LATC0 = 0;
                        LATCbits.LATC1 = 0;
                        LATAbits.LATA2 = 0;
                        LATAbits.LATA3 = 0;
                        break;
                    case LEFT: // In LEFT-Mode wechseln
                        if(mode == DISABLE)
                            XLCDClear();
                        mode = LEFT;
                        LATCbits.LATC0 = 0;
                        LATCbits.LATC1 = 1;
                        LATAbits.LATA2 = 0;
                        LATAbits.LATA3 = 1;
                        break;
                    case RIGHT: // In RIGHT-Mode wechseln
                        if(mode == DISABLE)
                            XLCDClear();
                        mode = RIGHT;
                        LATCbits.LATC0 = 1;
                        LATCbits.LATC1 = 0;
                        LATAbits.LATA2 = 1;
                        LATAbits.LATA3 = 0;
                        break;
                    case DISABLE: // In DISABLE-Mode wechseln
                        mode = DISABLE;
                        LATCbits.LATC0 = 1;
                        LATCbits.LATC1 = 1;
                        LATAbits.LATA2 = 1;
                        LATAbits.LATA3 = 1;

                        // In DISABLE-Mode alle bisher eingegebenen Zahlen verwerfen...
                        char_pointer = 0;
                        row = 0;

```

```

        col = 0;
        XLCDClear();
        XLCDGoto(0,0);
        XLCDPutROMString((ROM char*)"DISABLED!");
    }
}
else if(isdigit(data) && mode != DISABLE) // ..und keine mehr annehmen
{
    XLCDGoto(row, col); // Cursor Position wiederherstellen
    if(char_pointer == 0)
    {
        XLCDClear();
        XLCDGoto(0, 0);
    }
    if(char_pointer == CHAR_MAX)
    {
        XLCDGoto(0, 0);
        XLCDPutROMString((ROM char*)"buffer overflow");
        char_pointer = 0;
    }
    else
    {
        xlcd_buffer_higher[char_pointer++] = data;
        XLCDPut(data);
    }
    XLCDGetCursorPos(&row, &col); // Cursor Position wiederherstellen
}
else if(data == CR && mode != DISABLE) // Nach Druecken von Enter...
{
    if(char_pointer == 0)
    {
        XLCDGoto(0, 0);
        XLCDPutROMString((ROM char*)"no val assigned");
    }
    else
    {
        xlcd_buffer_higher[char_pointer] = 0; // ... Zahlenstring terminieren
        duty_cycle = (unsigned int)atoul(xlcd_buffer_higher); // und zuweisen
        if(duty_cycle > 1023)
        {
            XLCDGoto(0, 0);
            XLCDPutROMString((ROM char*)"val must be<1024");
            char_pointer = 0; // Puffer zuruecksetzen
        }
        else
        {
            SetDCPWM1(duty_cycle);
            XLCDGoto(0, 0);
            XLCDPutROMString((ROM char*)"value assigned");
            char_pointer = 0; // Puffer zuruecksetzen
        }
    }
}
else if(data == BS && mode != DISABLE) // Zeichen aus Puffer loeschen
{
    XLCDGetCursorPos(&row, &col); // Cursor Position wieder herstellen
    if(col > 0) // Falls mindestens ein Zeichen im Puffer
    {
        XLCDCursorLeft();
        data = SPACE; // Leerzeichen auf LCD ausgeben
        XLCDPut(data);
        XLCDCursorLeft();
        char_pointer--; // Pufferposition um eine Stell zuruecksetzen
    }
}
}
// Ausgabe der Drehgeschwindigkeit
ltoa(speed, xlcd_buffer_lower);
strcatpgm2ram(xlcd_buffer_lower, (ROM char*)" U/min    ");
XLCDGoto(1, 0);
XLCDPutString(xlcd_buffer_lower);
}
return 0;
}

```